Code Resources Reference

This section describes the data types and CE-provided routines you can use in a code resource (resource type `'detc'`). It also describes the routines that your code resource can provide and the circumstances in which the CE calls each of these routines.

Rules for Writing Code Resources

Because AOCE templates extend the Finder and are called by the Finder, it is possible for a code resource routine to corrupt the Finder or cause it to crash. To make sure that your code resource causes no problems, follow these rules:

- Use as little memory as possible. Try to allocate all the memory you need when you initialize the template (in your `kDETcmdInit` routine, page 5-150) and provide error handling for insufficient-memory cases whenever you allocate memory.
- Don't use global variables. The CE does not maintain an A5 world for template code resources. If your compiler uses global space for inline code, you must not use such code in your routines.
- Don't assume that the CE locks down your code resource. In the interval between calls by the CE to your code resource, your code is unlocked and purgeable. You cannot use callback or completion routines for operations that don't complete before they return to the CE. If you must use a completion or callback routine for a function that you call asynchronously, you must load your own code resource into memory and lock it. Note, however, that doing so interferes with the Finder's efficient use of memory, causing problems for the user.
- Before changing anything, always save the state of the system, including the graphics state, resource chain, and current file, and restore them before returning to the CE or calling any CE callback routines.

Data Types

The routines in an AOCE template code resource use the data types described in this section.

Target Specifier

Many routines in an AOCE template code resource refer to a specific aspect. The AOCE template target specifier specifies the aspect to which the routine applies. The target specifier is defined by the `DETTargetSpecification` structure.

```
struct DETTargetSpecification
{
      DETTargetSelector selector;   /* target selector */
      RStringPtr aspectName;        /* aspect name */
      long itemNumber;              /* sublist index number */
      PackedDSSpecPtr dsSpec;       /* DSSpec */
};
```

Field descriptions

`selector` A value that indicates whether the specified aspect is the current aspect (the one with which the code resource is associated) or some other aspect. The possible values for this field are listed following these field descriptions.

`aspectName` A pointer to the name of the aspect. You can specify `nil` for this field if the target is a main aspect and the value of the `selector` field is not `kDETSelf`. For target specifiers that the CE sends to your code resource, however, this field is always filled in if the target is an aspect, even if it's a main aspect. If you receive a target specifier with a `nil` in this field, the target is not an aspect (it might be a template, for example).

`itemNumber` If the value of the `selector` field is `kDETSublistItem`, then the `itemNumber` field contains the index number of an item in the current aspect's sublist. Item numbers start with 1. If the `selector` field is set to `kDETSelectedSublistItem`, then the index number counts only items in the sublist that the user has selected. If the `selector` field is set to `kDETAspectTemplate`, then the target is the aspect template indexed by the `itemNumber` field (the CE assigns an index number to every template that it loads into memory). If the `selector` field is set to `kDETInfoPageTemplate`, the target is the information page template indexed by the `itemNumber` field. If the `selector` field is set to any other value, the CE ignores this field.

`dsSpec` A pointer to a `DSSpec` structure. If the `selector` field is set to `kDETDSSpec`, then the `dsSpec` field indicates the target item. If the `selector` field is set to any other value, the CE ignores this field.

```
enum DETTargetSelector {
   kDETSelf = 0,              /* the current item */
   kDETSelfOtherAspect,       /* another aspect of the current item */
   kDETParent,                /* the parent of the current item */
   kDETSublistItem,           /* the ith item in the sublist */
   kDETSelectedSublistItem,   /* the ith selected item in the sublist */
   kDETDSSpec,                /* DSSpec */
   kDETAspectTemplate,        /* specific aspect template */
```

```
   kDETInfoPageTemplate,      /* specific info-page template */
   kDETHighSelector = 0xF000  /* force type to be short */
};

typedef enum DETTargetSelector DETTargetSelector;
```

Constant descriptions

`kDETSelf` The target aspect is the current one; that is, the aspect that originated the call to the code resource. The CE ignores all fields other than the `selector` field. When the CE calls your code resource to handle a targeted event, it sets the target selector type to `kDETSelf`. If your code resource doesn't handle the event and the aspect is an attribute, the CE calls the aspect's parent record and sets the selector type to `kDETSublistItem`.

`kDETSelfOtherAspect` The target is another aspect of the record or attribute to which the current aspect applies. The `aspectName` field points to the name of the target aspect.

`kDETParent` The target is an aspect of the object in whose sublist the current object resides. That is, the current aspect is for an attribute, and the target aspect is an aspect of the record that contains that attribute. The `aspectName` field points to the name of the target aspect, which can be any aspect of the parent.

`kDETSublistItem` The target is an aspect of an item in the sublist of the current aspect. The `itemNumber` field contains the index number of the item in the sublist. Index numbers start with 1. The `aspectName` field points to the name of the target aspect. When you call a routine provided by the CE, you can set the `aspectName` field to `nil` to target the main aspect. This selector type is useful for iterating through all of the items in a sublist. When the CE calls your code resource to handle a targeted event, it sets the target selector type to `kDETSelf`. If your code resource doesn't handle the event, the CE calls the aspect's parent and sets the selector type to `kDETSublistItem`.

`kDETSelectedSublistItem` The target is an aspect of an item in the sublist of the current aspect. The `itemNumber` field contains the index number of the item in the sublist, counting only the items the user has selected. Index numbers start with 1. The `aspectName` field points to the name of the target aspect. When you call a routine provided by the CE, you can set the `aspectName` field to `nil` to target the main aspect. This selector type is useful for iterating through all of the items that the user has selected in a sublist.

`kDETDSSpec` The target is the item specified by the `dsSpec` field. You must wait until the `kDETPastFirstLookup` metaproperty changes to 1 before you can target a catalog object. Metaproperties are listed in Table 5-3 on page 5-86.

`kDETAspectTemplate`
: The target is the aspect template indexed by the `itemNumber` field. The CE assigns an index number to every aspect template that it loads into memory. You can use this target selector only with the callback routines `kDETcmdGetResource` (page 5-207) and `kDETcmdGetTemplateFSSpec` (page 5-206).

`kDETInfoPageTemplate`
: The target is the information page template indexed by the `itemNumber` field. The CE assigns an index number to every information page template that it loads into memory. You can use this target selector only with the callback routines `kDETcmdGetResource` (page 5-207) and `kDETcmdGetTemplateFSSpec` (page 5-206).

Forwarder List

Your `kDETcmdDynamicForwarders` code-resource routine (page 5-155) returns a linked list of forwarder items, each of which contains the same information as a forwarder template (see "Components of Forwarder Templates" beginning on page 5-138). A forwarder item is defined by the `DETForwarderListItem` structure.

```
struct DETForwarderListItem {
   struct DETForwarderListItem** next;/* handle to next item, or nil */
   AttributeTag attributeValueTag;    /* attribute value tag (0 for none) */
   PackedPathName rstrs;              /* forwarder list */
};
```

The `rstrs` field is a list of packed `RString` structures in the format defined by the `PackedPathName` data type. This field contains the record type (an empty, or zero-length, string if none), the attribute type (empty if none), and a list of template names to forward to. The `PackedPathName` data type and functions for working with `PackedPathName` and `RString` structures are defined in the chapter "AOCE Utilities" in this book.

Call Block Headers

When the Catalogs Extension calls your code resource, it passes it a pointer to an AOCE template call block. The call block indicates which event occurred and includes additional parameters specific to each type of event. Every call block starts with the same fields, described here. The fields specific to each event are listed and described with the description of the code-resource routine that you must provide to handle the event. See "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148 for these descriptions.

There are three headers for call blocks: the AOCE template call block header, the AOCE template call block targeted header, and the AOCE template call block property header. These headers all have several fields in common. All of the fields are described in this section following the header definitions.

```
#define DETCallBlockHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */

#define DETCallBlockTargetedHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */

#define DETCallBlockPropertyHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */\
   short property;               /* the property number the call refers to */
```

Field descriptions

`reqFunction` A routine selector that tells you which of your code resource routines to execute. For a list of the routine selectors and a description of the routines, see "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148.

`callBack` A pointer to the CE's entry point for CE routines that you can call from your code resource. If you want to call one of the CE's callback routines, pass the parameters described with that routine to the routine at the address in this field. You can use the `CallBackDET` macro described on page 5-197 for this purpose. The available AOCE template callback routines are described in "CE-Provided Functions That Your Code Resource Can Call" starting on page 5-196.

`callBackPrivate` Reserved.

`templatePrivate`
Private storage for use by the code resource. You provide a value for this field when the code resource first calls your `kDETcmdInit` routine. The CE saves this value until you execute your `kDETcmdExit` routine and includes it in the parameter block every time it calls the code for any aspect created from this aspect template. Your code resource can change this value at any time.

`instancePrivate`
Private storage for use by the code resource. The CE maintains a separate `instancePrivate` field for each instance of an aspect template; that is, for each aspect. You provide a value for this field when the code resource first calls your `kDETcmdInstanceInit` routine. The CE saves this value until you execute your `kDETcmdInstanceExit` routine and includes it in the parameter block every time it calls the code for this aspect. Your code resource can change this value at any time.

`target` A target specifier structure indicating which aspect was the original target of the event. For example, if the CE calls the code resource for an attribute and that code resource doesn't handle the event, the CE calls the code resource for the record that contains the attribute. In that case, the target specifier identifies the attribute that was called initially. See "Target Specifier" on page 5-142.

`targetIsMainAspect`
A Boolean value that indicates whether the target is a main aspect (`true`) or not (`false`).

`property` The property number of the property the routine refers to.

Callback Block Headers

When your code resource calls a function supplied by the Catalogs Extension, the code resource passes a pointer to an AOCE template callback block. The callback block indicates which routine it wants the CE to execute and includes additional parameters specific to each type of routine. Every callback block starts with the same fields, described here. The fields specific to each routine are listed and described with the description of the callback routine. See "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196 for these descriptions.

There are three headers for callback blocks: the AOCE template callback block header, the AOCE template callback block targeted header, and the AOCE template callback block property header.

```
#define DETCallBackBlockHeader \
   DETCallBackFunctions reqFunction;   /* requested function */
```

```
#define DETCallBackBlockTargetedHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */

#define DETCallBackBlockPropertyHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */\
   short property;                     /* the property to apply the
                                          request to */
```

Functions You Can Provide as Part of Your Code Resource

The AOCE Catalogs Extension calls your code resource when certain events occur, such as a change in an attribute value or a mouse-down event in a custom view. Your code resource must be reentrant. The CE might call the routines in your code resource at any time and in any order (except for a few routines, such as your initialization and exit routines, as indicated in this chapter).

If your code resource does not handle an event, it must return the `kDETDidNotHandle` result code. If it successfully handles the event, your code resource should return the `noErr` result code. You can return any negative number as a result code to indicate an error.

If an attribute does not have a code resource, or your code resource for the attribute doesn't handle an event, the CE calls the code resource (if any) in the aspect for the record that is the parent of the code resource it called originally.

The CE passes to your code resource a pointer to a parameter block. The fields of the parameter block are described in the preceding section and in the individual routine descriptions in this section. The function prototype for your code resource's main routine is

```
pascal OSErr MyCode(DETCallBlockPtr callBlockPtr);
```

The `DETCallBlock` structure is a union of all the parameter blocks for the code resource routines. The routine selector is specified by the `reqFunction` field of the parameter block header. You can read this field as follows:

```
callBlockPtr->protoCall.reqFunction
```

IMPORTANT

The CE does not save your code resource's A5 world. You cannot use application global variables in your code resource. ▲

Call-For Mask

Most code resources do not need to respond to the majority of events for which the Catalogs Extension can call your code resource. To avoid being called unnecessarily, each template's code resource has a "call-for" mask that indicates the events for which it should be invoked. Your code-resource initialization routine must return the call-for mask when it is called for initialization. In addition, your code resource can use the `kDETcmdChangeCallFors` callback routine (page 5-198) to change the call-for mask.

Not every possible event has a corresponding bit in the call-for mask. There are two classes of events excepted from the call-for mask: events that occur very infrequently (such as initialization), and events that occur only because the template specifically caused them (for instance, by including a custom view in an information page view list). Your code resource is always called for all such events unless you specify a value of `kDETCallForNothing` for your call-for mask. To be called *only* for such events, specify `kDETCallForOther`.

A parent object is not given calls that its children failed to handle unless the `kDETCallForEscalation` bit is set in the call-for mask.

```
/* Call-for list: */

#define kDETCallForOther         1     /* call for events not listed below */
#define kDETCallForIdle          2     /* kDETcmdIdle */
#define kDETCallForCommands      4     /* kDETcmdPropertyCommand,
                                          kDETcmdSelfOpen */
#define kDETCallForViewChanges   8     /* kDETcmdViewListChanged,
                                          kDETcmdPropertyDirtied,
                                          kDETcmdMaximumTextLength */
#define kDETCallForDrops         0x10  /* kDETcmdDropQuery,
                                          kDETcmdDropMeQuery */
#define kDETCallForAttributes    0x20  /* kDETcmdAttributeCreation,
                                          kDETcmdAttributeNew,
                                          kDETcmdAttributeChange,
                                          kDETcmdAttributeDelete */
#define kDETCallForValidation    0x40  /* kDETcmdValidateSave */
#define kDETCallForKeyPresses    0x80  /* kDETcmdKeyPress and
                                          kDETcmdPaste */
#define kDETCallForSyncing       0x200 /* kDETcmdShouldSync, kDETcmdDoSync */
#define kDETCallForResources     0x100 /* kDETcmdDynamicResource */
#define kDETCallForEscalation    0x8000/* all calls escalated to the
                                          next level */

#define kDETCallForNothing       0     /* do not call */
#define kDETCallForEverything    0xFFFFFFFF  /* all of the above */
```

Initializing and Removing Templates

The Catalogs Extension calls the code resource routines in this section when it loads aspect templates into memory (`kDETcmdInit`), creates aspects (`kDETcmdInstanceInit`), creates attributes or records (`kDETcmdItemNew`), removes an aspect template from memory (`kDETcmdExit`), and removes an aspect from memory (`kDETcmdInstanceExit`).

kDETcmdInit

The CE calls your code resource with this routine selector when the CE first loads the template.

```
struct DETInitBlock {
            DETCallBlockHeader
            long newCallFors;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdInit`
↔	`templatePrivate`	`long`	Data stored in template
→	`callBack`	`DETCallBack`	Callback pointer
←	`newCallFors`	`long`	Call-for list

DESCRIPTION

The Catalogs Extension calls your code resource with the `kDETcmdInit` routine selector only when the Finder loads your aspect template during template initialization (such as during system initialization or the first time the CE needs a template after you have called the `kDETcmdUnloadTemplates` callback routine). You should use this opportunity to initialize the call-for mask for your template and to allocate any memory your template needs. Return the call-for mask in the `newCallFors` field. Place a pointer to your template's data in the `templatePrivate` field of the parameter block.

You can call the CE callback routines `kDETcmdGetTemplateFSSpec`, `kDETcmdBeep`, or `kDETcmdAboutToTalk`. You should use the routine `kDETcmdAboutToTalk` only if you need to report a problem to the user.

Return the `noErr` result code if you return a new call-for list. If you set the call-for mask to `kDETCallForEverything`, return the `kDETDidNotHandle` result code. If for some reason you do not want the template to be loaded (for example, if you cannot allocate the memory you need), return an error code.

SPECIAL CONSIDERATIONS

Because the CE has not yet created any aspects, you cannot call any targeted callback routines from your initialization routine.

Because the CE might not have loaded all main aspect templates, the Standard Catalog Package does not yet have information on the icons, record types, and record categories available. Therefore, your initialization routine cannot call the Standard Catalog Package functions `SDPGetIconByType`, `SDPGetDSSpecIcon`, `SDPGetCategories`, and `SDPGetCategoryTypes`.

Because the Collaboration toolbox might not yet be available, do not call any Collaboration toolbox functions unless you have used the Gestalt Manager to check for its availability.

CALL-FOR MASK VALUE

None

SEE ALSO

Call-for masks are described in "Call-For Mask" on page 5-149.

All Standard Catalog Package functions are described in the chapter "Standard Catalog Package" in this book.

kDETcmdExit

The CE calls your code resource with this routine selector before it removes the template.

```
struct DETExitBlock{
            DETCallBlockHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdExit`
→	`templatePrivate`	`long`	Data stored in template
→	`callBack`	`DETCallBack`	Callback pointer

DESCRIPTION

The Catalogs Extension calls your exit routine just before it removes your template. The CE removes templates when the system shuts down or when you call the `kDETcmdUnloadTemplates` callback routine. Your exit routine should free any memory you allocated. You can call the CE callback routines `kDETcmdGetTemplateFSSpec`, `kDETcmdBeep`, or `kDETcmdAboutToTalk`. You

should use the routine `kDETcmdAboutToTalk` only if you need to report a problem to the user.

Your exit routine should return the `noErr` or `kDETDidNotHandle` result code or a specific error code.

SPECIAL CONSIDERATIONS

Because the AOCE toolbox might have already been shut down, your exit routine should not call any AOCE functions.

CALL-FOR MASK VALUE

None

kDETcmdInstanceInit

The CE calls your code resource with this routine selector when it creates an aspect from your aspect template.

```
struct DETInstanceInitBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdInstanceInit`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

The Catalogs Extension calls your instance-initialization routine once each time it creates an aspect from your aspect template. You should allocate any memory needed by this aspect and place a pointer to the aspect's data in the `instancePrivate` field of the parameter block.

If your routine returns an error (any negative result code), the CE disables your code resource and does not call it again for this aspect. In all other respects the aspect continues to function normally, and the CE can still call your code resource for other aspects for the same template.

If your routine returns either the `noErr` or `kDETDidNotHandle` result codes, the CE processes the aspect normally.

The CE can remove this aspect from memory at any time that the aspect is not in use and the Finder needs the memory. In that case, the CE calls your `kDETcmdInstanceExit` routine. The CE will then call your `kDETcmdInstanceInit` routine again whenever it needs the aspect, such as when the user opens a record or causes a catalog folder window to redraw.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceExit` routine is described on page 5-154.

kDETcmdItemNew

The CE calls your code resource with this routine selector when it creates a new record or attribute.

```
struct DETItemNewBlock{
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdItemNew`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

After the Catalogs Extension creates an aspect and calls your `kDETcmdInstanceInit` routine, the CE calls your new item routine each time it creates a new record or attribute. You can use this opportunity to specify initial values for attributes or perform other actions appropriate to a new attribute or record of the type supported by this aspect.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceInit` routine is described on page 5-152.

kDETcmdInstanceExit

The CE calls your code resource with this routine selector before it removes an aspect from memory.

```
struct DETInstanceExitBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdInstanceExit`
↔	`templatePrivate`	`long`	Data stored in template
→	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

The Catalogs Extension can remove an aspect from memory at any time the aspect is not in use and the Finder needs additional memory. Your instance exit routine should release any memory allocated by the `kDETcmdInstanceInit` routine for this aspect. The CE ignores the result code your instance exit routine returns.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceInit` routine is described on page 5-152.

Dynamic Creation of Resources

Your code resource can extend the use of your templates much as a forwarder template does. Your code resource can also substitute resources for those in the template file. Because the Catalogs Extension loads resources as needed, it can call your code resource at any time for this purpose. The CE calls the code resource routines described in this section to achieve these ends.

kDETcmdDynamicForwarders

The CE calls your code resource with this routine selector to allow you to apply the template to additional record or attribute types.

```
struct DETDynamicForwardersBlock {
            DETCallBlockHeader
            DETForwarderListHandle forwarders;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdDynamicForwarders`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
←	`forwarders`	`DETForwarderListHandle`	List of forwarders

DESCRIPTION

When the Catalogs Extension is loading your template, after it calls your `kDETcmdInit` routine, it calls your `kDETcmdDynamicForwarders` routine to allow you to add record or attribute types to those to which the template applies. The `forwarders` field is a handle to a linked list of elements of type `DETForwarderListItem`. Each contains the same information as is found in a forwarder template: a record type, attribute type, or both; an attribute value tag (0 for none); and a list of template names (including both aspect and information page templates).

Your `kDETcmdDynamicForwarders` routine must allocate the handles containing the returned data, but the CE disposes of them when done.

If your routine returns `kDETDidNotHandle` or an error, the CE does not process the forwarders list.

CALL-FOR MASK VALUE

None

SEE ALSO

The `DETForwarderListItem` structure is defined in "Forwarder List" on page 5-145.

The forwarder template is described in "Components of Forwarder Templates" beginning on page 5-138.

kDETcmdDynamicResource

The CE calls your code resource with this routine selector when it is about to load a resource from your template file to give you the opportunity to substitute a different resource.

```
struct DETDynamicResourceBlock {
            DETCallBlockTargetedHeader
            ResType resourceType;
            short propertyNumber;
            short resourceID;
            Handle theResource;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdDynamicResource`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`resourceType`	`ResType`	Type of resource being requested
→	`propertyNumber`	`short`	Property number of requested resource
→	`resourceID`	`short`	Resource ID
←	`theResource`	`Handle`	Replacement resource

DESCRIPTION

Before the Catalogs Extension loads a resource from your template file, it calls your `kDETcmdDynamicResource` routine to give you the opportunity to substitute a different resource for the one in the file. The CE calls this routine for any resource except the aspect template signature resource (`'deta'`) and those used by a forwarder template or the `kDETcmdDynamicForwarders` routine (the attribute value tag, attribute type, and record type resources; see Table 5-11 on page 5-138.)

The `resourceType` field contains the type of resource required. The `propertyNumber` field contains the property number of the resource, and the `resourceID` field contains the resource ID of the resource (that is, the base template resource ID plus the property number).

If you want to substitute a different resource for the one in the template file, return a handle to the new resource in the `theResource` field. You must allocate the handle; the CE disposes of it when finished with it.

If your routine returns `kDETDidNotHandle` or an error, then the CE uses the resource from the template file.

SPECIAL CONSIDERATIONS

Do not allocate a resource handle for the `theResource` field; you must own this handle.

Because the CE calls your `kDETcmdDynamicResource` routine every time it loads a resource, you should include such a routine only if you have a specific reason to do so. Otherwise, use the call-for mask to avoid having the CE call your code resource for resource loading.

CALL-FOR MASK VALUE

`kDETCallForResources`

SEE ALSO

Before loading resources from a forwarder template file, the CE calls your `kDETcmdDynamicForwarders` routine (page 5-155).

Processing Idle-Time Tasks

When an information page that uses your aspect template is the frontmost window, the CE calls your code resource periodically with the `kDETcmdIdle` routine selector.

kDETcmdIdle

The CE calls your code resource with this routine selector periodically during idle times.

```
struct DETInstanceIdleBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdIdle`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

The Catalogs Extension calls your code resource with this routine selector during idle times when an information page that uses your aspect template is the Finder's frontmost window and the Finder is the frontmost application. An aspect code resource cannot perform an idle-time task unless its window is frontmost.

The CE ignores the result code returned by this routine. Therefore, the CE does not call the parent record's code resource when the aspect for an attribute returns `kDETDidNotHandle`.

CALL-FOR MASK VALUE

`kDETCallForIdle`

Property and Information Page Functions

The routines in this section interact directly with an information page. The Catalogs Extension calls your `kDETcmdOpenSelf` routine, described next, to give you the opportunity to override the standard behavior when the user opens an information page. Your `kDETcmdPropertyCommand` routine (page 5-159) processes a command sent by an information page property, such as a button or menu item. Your `kDETcmdKeyPress` (page 5-163) and `kDETcmdPaste` (page 5-164) routines handle keypresses and paste operations that occur when the user is using your information page.

Your `kDETcmdMaximumTextLength` routine (page 5-166) specifies the maximum permitted length for a text string in an information page.

The CE calls your `kDETcmdViewListChanged` routine (page 5-166) when the list of enabled views has changed in an information page. The CE calls your `kDETcmdPropertyDirtied` routine (page 5-167) to give you an opportunity to update the information page display when a property value changes. The CE calls your `kDETcmdValidateSave` routine (page 5-168) when the CE is about to save property values.

kDETcmdOpenSelf

The CE calls your code resource with this routine selector before it opens an information page to give you the opportunity to override the normal behavior.

```
struct DETOpenSelfBlock {
            DETCallBlockTargetedHeader
            short modifiers;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdOpenSelf`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`modifiers`	`short`	Modifier keys

DESCRIPTION

When a user attempts to open a catalog object for which you provided the main aspect, the Catalogs Extension calls the code resource of that main aspect with the `kDETcmdOpenSelf` routine selector before opening the information page. You can use this opportunity to do something other than opening the information page or to set default values for the information page before it opens.

Because the target is always a main aspect when you receive this routine selector, the value of `targetIsMainAspect` is always `true`.

If your routine returns the `kDETDidNotHandle` result code, the CE opens the information page normally. If it returns `noErr`, the CE does not open the information page. If it returns an error, the CE displays a Finder error dialog box and does not open the information page.

CALL-FOR MASK VALUE

`kDETCallForCommands`

kDETcmdPropertyCommand

The CE calls your code resource with this routine selector when the user takes certain actions in an information page.

```
struct DETPropertyCommandBlock {
            DETCallBlockPropertyHeader
            long parameter;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPropertyCommand
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
↔	parameter	long	Command parameter

DESCRIPTION

The Catalogs Extension calls your property-command routine when the user clicks a button or checkbox in an information page, selects an item in a pop-up menu, or clicks on static command text. The CE also calls your property-command routine when you return a property number in response to a drop operation that affects your aspect or when you call the `kDETcmdDoPropertyCommand` callback command.

When it calls your property command, the CE always includes a value in the `property` field of the parameter block. This is always a number you have supplied; either in the command field of the `'detv'` resource for the view that originated the property command, or as a parameter to your `kDETcmdDropQuery` routine or the `kDETcmdDoPropertyCommand` callback routine. Most commonly, you use the property number of the view as the value of the command field of the `'detv'` resource so that your code resource can tell which view sent the property command. Some property commands, such as those related to drop operations, do not originate from views. In this case, you use the value of the `property` field of the parameter block as a routine selector rather than as a property number.

Some property commands include a parameter in the `parameter` field of the parameter block; for example, if the user selects an item in a pop-up menu, the CE includes the number of the menu item in the `parameter` field of the parameter block when it calls your property-command routine. Table 5-14 shows the various property commands that you can handle in your code resource and describes the origin of the value in the `property` and `parameter` fields of the parameter block for each type of property command.If your routine returns a result code of `noErr`, the CE assumes that you have handled the command and takes no further action. If your routine returns a result code of `kDETDidNotHandle`, the CE calls the code resource of the parent of the object whose code resource was called originally (that is, if the code resource was for an attribute, the CE calls the code resource, if any, of the record that contains that attribute). If your routine returns a value in the `parameter` field when it returns a result code of `kDETDidNotHandle` for radio buttons, checkboxes, and pop-up menus, the CE sets the value of the property equal to the value in the `parameter` field.

If your routine returns an error code (any nonzero result code), the CE displays a dialog box specifying the error and leaves the value of the property unchanged.

Table 5-14 Property commands

Source of command	`property` field of parameter block	`parameter` field of parameter block
Button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Not used.
	NOTE If your routine returns `kDETDidNotHandle`, the CE ignores the button click. If you use the property numbers `kDETAddNewItem`, `kDETRemoveSelectedItems`, or `kDETOpenSelectedItems`, the CE handles the command without calling your code resource (see Table 5-3 on page 5-86).	
Default button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Not used.
	NOTE If your routine returns `kDETDidNotHandle`, the CE ignores the button click.	
Radio button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Value in the command-parameter field of the `'detv'` resource for the button. Each button in a set of radio buttons must have a distinct value.
	NOTE The button displayed as "on" is the one for which the command-parameter field is equal to the value of the property. If your property-command routine returns `kDETDidNotHandle`, the CE sets the value of the property equal to the value in the `parameter` field of the parameter block. Therefore, if you do not alter the value in the `parameter` field, the button the user clicked is displayed as "on." If you do handle the radio-button command yourself, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the radio button.	
Checkbox	Value in the property-command field of the `'detv'` resource for the checkbox; this must equal the property number of the checkbox.	Set by the CE to the opposite of the current property value (that is, 1 if the property value is 0, or 0 if the property value is 1).
	NOTE The property value for a checkbox can be equal to 0 or 1; the checkbox is off if this value is 0 and on if 1. If your code resource returns `kDETDidNotHandle`, the CE sets the property value to equal the value in the `parameter` field, toggling the checkbox off or on. If you do handle the checkbox command yourself, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the checkbox.	

continued

Table 5-14 Property commands (continued)

Source of command	`property` field of parameter block	`parameter` field of parameter block
Pop-up menu	Value in the property-command field of the `'detv'` resource for the menu; this must equal the property number of the menu.	Value in the command-ID field of the `'fmnu'` resource that defines the menu. There is a distinct command-ID value for each menu item.
	NOTE Your code resource can use the `command` parameter to determine which item in the pop-up menu the user has chosen.	
Static command text from view	Value in the property-command field of the `'detv'` resource for the view.	Value in the command-parameter field of the `'detv'` resource for the view.
	NOTE If you use the value `kDETChangeViewCommand` for the property-command field of the `'detv'` resource, the CE uses this property command to sort a sublist and does not call your code resource.	
Drop-operation command	Value you specified in the `commandID` parameter to your `kDETcmdDropQuery` routine. Treat this value as a routine selector to determine what course of action to take.	The location of the cursor when the mouse button was released, in global coordinates, as two shorts in the order x, y.
	NOTE When the user drops one or more objects on a catalog object for which you have provided an aspect template, the CE calls your `kDETcmdDropQuery` routine once for each item dropped. If your routine returns a property number, the CE calls your property-command routine. The CE combines all of the drop operations that return the same property number and calls your property command only once. You can then call the `kDETcmdGetCommandSelectionCount` callback routine to determine how many objects are being dropped and the `kDETcmdGetCommandItemN` callback routine to determine the nature of each object being dropped.	
Drop-me operation command	Value you specified in the `commandID` parameter to your `kDETcmdDropMeQuery` routine. Treat this value as a routine selector to determine what course of action to take.	The location of the cursor when the mouse button was released, in global coordinates, as two shorts in the order x, y.
	NOTE When the user drags and drops a catalog object for which you have provided an aspect template, the CE calls your `kDETcmdDropMeQuery` routine. If your routine returns a property number, the CE calls your property-command routine. You can then call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object upon which the item is being dropped.	

Table 5-14 Property commands (continued)

Source of command	`property` field of parameter block	`parameter` field of parameter block
Do-property-command callback	Value you specified in the `property` parameter to the `kDETcmdDoPropertyCommand` callback routine.	Value you specified in the `parameter` parameter to the `kDETcmdDoPropertyCommand` callback routine.

NOTE The `kDETcmdDoPropertyCommand` callback routine allows your code resource to send a property command to any code resource you can target.

CALL-FOR MASK VALUE

None

SEE ALSO

To force the CE to redraw a view after you handle a property event, use the `kDETcmdDirtyProperty` callback routine (page 5-233).

The aspect's `kDETAspectViewMenu` resource is described on page 5-103.

View types are described on page 5-127.

The `kDETcmdDropMeQuery` routine is described on page 5-170. The `kDETcmdDropQuery` routine is described on page 5-172.

Use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped and the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of each object dropped.

To initiate a property command from a code resource, use the `kDETcmdDoPropertyCommand` routine (page 5-245).

kDETcmdKeyPress

The CE calls your code resource with this routine selector when the user presses a key while using an information page.

```
struct DETKeyPressBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdKeyPress
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	theEvent	EventRecord	The event record for the keypress

DESCRIPTION

You can use your `kDETcmdKeyPress` routine to respond to a keypress that occurs while the user is using your information page. If the user is editing a text view, the `property` field identifies the view. If the cursor is not in a text view, the `property` field contains the value `kDETNoProperty`. The Catalogs Extension does not call your `kDETcmdKeyPress` routine for Command-key keypress combinations.

If your routine returns `kDETDidNotHandle`, the CE handles the keypress. If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr`, the CE assumes you handled the keypress and does no further processing.

You can use this routine, for example, to prevent the user from entering certain characters in a text field.

CALL-FOR MASK VALUE

None

SEE ALSO

To control what a user pastes into your information page, use the `kDETcmdPaste` routine, described next.

kDETcmdPaste

The CE calls your code resource with this routine selector when the user attempts to paste text while using your information page.

```
struct DETPasteBlock {
            DETCallBlockPropertyHeader
            short modifiers;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdPaste`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`modifiers`	`short`	Modifier keys at time of paste

DESCRIPTION

The Catalogs Extension calls your `kDETcmdPaste` routine when the user chooses Paste from the Edit menu (or presses Command-V when Paste is enabled) while the user is using your information page. If the user is editing a text view, the `property` field identifies the view. If the cursor is not in a text view, the `property` field contains the value `kDETNoProperty`.

If your routine returns `kDETDidNotHandle`, the CE handles the paste. If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr`, the CE assumes you handled the paste and does no further processing.

You can use this routine, for example, to prevent the user from pasting certain characters in a text field.

CALL-FOR MASK VALUE

None

SEE ALSO

To determine what the user is attempting to paste you must read the data in the scrap. For information on how to read the scrap, see the chapter "Scrap Manager" in *Inside Macintosh: More Macintosh Toolbox*.

kDETcmdMaximumTextLength

The CE calls your code resource with this routine selector to determine the maximum permitted length of a property that is displayed in an editable text view.

```
struct DETMaximumTextLengthBlock {
            DETCallBlockPropertyHeader
            long maxSize;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdMaximumTextLength`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
←	`maxSize`	`long`	Maximum text length

DESCRIPTION

If the user tries to type more into an editable text view than the maximum you specify with your `kDETcmdMaximumTextLength` routine, the Catalogs Extension displays a dialog box informing the user that the text has reached its maximum length. The maximum size you can specify in the `maxSize` field is 255 bytes. When counting the length of a text string for this routine, count the first byte as 1, not as 0. If your routine returns an error or a result code of `kDETDidNotHandle`, the CE limits the text string to 255 bytes. If your routine returns `noErr`, the CE limits the text string to the length you specify.

CALL-FOR MASK VALUE

`kDETCallForViewChanges`

kDETcmdViewListChanged

The CE calls your code resource with this routine selector when the list of enabled views has changed in one of the information pages associated with this aspect.

```
struct DETViewListChangedBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdViewListChanged
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?

DESCRIPTION

The list of enabled views in an information page changes when the Catalogs Extension displays a conditional view. The list also changes when the page is first opened, either because the record has just been opened or because the user has used the pop-up menu to select a different information page.

CALL-FOR MASK VALUE

kDETCallForViewChanges

SEE ALSO

Conditional views are described in "Conditional Views" on page 5-26 and in "Information Page Template Signature Resource" beginning on page 5-121.

kDETcmdPropertyDirtied

The CE calls your code resource with this routine selector when you call the kDETcmdDirtyProperty callback routine and when the kDETPastFirstLookup metaproperty changes.

```
struct DETPropertyDirtiedBlock {
            DETCallBlockPropertyHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPropertyDirtied
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number

DESCRIPTION

The Catalogs Extension calls your `kDETcmdPropertyDirtied` routine when you call the `kDETcmdDirtyProperty` callback routine to indicate that a property value has changed, requiring a view to be redrawn. The CE also calls this routine when the CE completes its first catalog lookup and the `kDETPastFirstLookup` metaproperty changes to 1. Although the CE updates the display when you call the `kDETcmdDirtyProperty` callback routine and when it completes a catalog search, you might want to redraw other property views that are dependent on the one that changed initially. Also, if your routine returns the `kDETDidNotHandle` result code when the CE calls your code resource for an attribute with the `kDETcmdPropertyDirtied` routine selector, the CE calls the code resource for the record that contains that attribute. You can use this technique to inform a parent of a change that occurred in a child. The CE ignores any other function results of this routine.

CALL-FOR MASK VALUE

`kDETCallForViewChanges`

SEE ALSO

The `kDETcmdDirtyProperty` callback routine is described on page 5-233.

Metaproperties are listed in Table 5-3 on page 5-86.

kDETcmdValidateSave

The CE calls your code resource with this routine selector when the CE is about to save the property values associated with an aspect.

```
struct DETValidateSaveBlock {
            DETCallBlockTargetedHeader
            RStringHandle errorString;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdValidateSave`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
←	`errorString`	`RStringHandle`	Handle to error string

DESCRIPTION

If you wish to allow the Catalogs Extension to save the new data entered into an information page, return a result code of `noErr` or `kDETDidNotHandle`. If you do not want the CE to save the data, return an error (a negative result code) and, in the `errorString` parameter, specify a handle to an error string telling why the information page should not be saved. You must allocate the handle to the error string; the CE deallocates it. The CE displays the error string in a dialog box to inform the user why the data could not be saved.

Normally, the CE saves new property values only when the user leaves the aspect; that is, when the user closes the information page or flips to another information page that uses a different aspect. You can call the `kDETcmdSaveProperty` command to save a property value at any time.

The CE does not call your `kDETcmdValidateSave` routine when someone changes a sublist field or the name of a stand-alone attribute, or when your code resource calls the `kDETcmdSaveProperty` command.

CALL-FOR MASK VALUE

`kDETCallForValidation`

SEE ALSO

Call the `kDETcmdSaveProperty` command (page 5-234) to save a property value.

Supporting Drops

If the standard drop-operation resources are not adequate for your needs, you can provide a code-resource routine to handle drops. You can write a `kDETcmdDropMeQuery` routine for the aspect template of the object being dropped and a `kDETcmdDropQuery` routine for the aspect template of the destination for the drop. The Catalogs Extension calls the code resource of the object being dropped first and then the code resource of the destination. Thus, the code resource of the destination can override that of the object being dropped.

Drags and drops are described in “Drags and Drops” on page 5-28 and drop-operation resources are described in “Supporting Drags and Drops” beginning on page 5-98.

kDETcmdDropMeQuery

The CE calls your code resource with this routine selector when the user attempts to drop the object to which your aspect template applies onto another object.

```
struct DETDropMeQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType;
            Boolean copyToHFS;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdDropMeQuery`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`modifiers`	`short`	Modifier keys at drop time
↔	`commandID`	`long`	Command ID
↔	`destinationType`	`AttributeType`	Attribute type of new attribute
←	`copyToHFS`	`Boolean`	Copy to HFS?

DESCRIPTION

When the user drags an AOCE catalog object and drops it onto another catalog object or onto an HFS object, the Catalogs Extension calls the code resource in the aspect of the dragged object with the `kDETcmdDropMeQuery` routine selector. (If a dragged attribute's aspect does not contain a code resource or if its code resource returns the `kDETDidNotHandle` result code, the CE calls the code resource of the aspect for the record that contains that attribute.) You can call the `kDETcmdGetCommandItemN` callback routine to get information about the destination object.

The `modifiers` field indicates which modifier keys, if any, the user was pressing when the mouse button was released.

The `commandID` and `destinationType` fields contain the CE's best guess as to the correct drop action. Possible values for the `commandID` parameter are as follows:

```
#define kDETDoNothing   'xxx0'
#define kDETMove        'move'
#define kDETDrag        'drag'
#define kDETAlias       'alis'
```

Constant descriptions

`kDETDoNothing`	Do nothing. The CE has no standard behavior for a drop of this sort. For example, the user might try dragging an attribute from a sublist and dropping it onto an application.
`kDETMove`	Move the object to a new location. For example, if the user drags an attribute from one sublist to another on the same volume, the CE's default behavior is to change the location of the attribute.
`kDETDrag`	Make a copy of the object. For example, if the user drags an attribute from a sublist on one volume to a sublist on another volume, the CE copies the attribute.
`kDETAlias`	Make an alias to the object. For example, if the user drags a record from a catalog folder and drops it onto another record, the CE creates an alias to the record and places it in an attribute in the destination record.

The `destinationType` field indicates the attribute type of the attribute that the CE creates as a result of the drop if the CE copies an attribute or creates an alias to a record.

If your routine returns a result code of `kDETDidNotHandle` or an error, the CE continues to try to determine the appropriate action. If the dragged object is an attribute, the CE looks for a code resource in an aspect of the parent record. Then the CE looks for code resources in the aspects of the destination object and in the parent of the destination object, if any.

If you wish, you can set new values for the `commandID` and `destinationType` fields and return a result code of `noErr`. The CE then uses the values you set for these parameters as input to any other code resources it finds. If no other aspect or code resource overrides these values, the CE carries out the action you specified. If the CE can't carry out the specified action, it displays a dialog box describing the problem.

You can also specify a number in the developers' property-number range (that is, `kDETFirstDevProperty` through 249) for the `commandID` parameter. In this case, the CE sends a property command (`kDETcmdPropertyCommand`) with that number to the target aspect (that is, the code resource of the aspect that the CE indicated as the target when it called your code resource). Your property-command routine can then call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the destination object.

If you set the `copyToHFS` parameter to `true`, the CE displays a dialog box asking the user to copy the object to the desktop (that is, to create an HFS version of the object) before performing the operation. For example, if the item is a record and you set the `copyToHFS` parameter to `true`, the CE asks the user to create an information card before performing the operation. Your property-command routine can use the `kDETcmdGetCommandItemN` callback routine to get the file system specification (`FSSpec` structure) for the information card. If you set the `copyToHFS` parameter to `true`, you must set the `commandID` parameter to a property number; otherwise, the CE ignores the `copyToHFS` parameter.

Note
In future versions of the AOCE software, the CE might create the HFS version of the object rather than requesting the user to do so. ◆

CALL-FOR MASK VALUE

`kDETCallForDrops`

SEE ALSO

For more information on how the CE handles drags and drops, see "Drags and Drops" on page 5-28.

If the object on which the AOCE catalog object was dropped is also a catalog object, the CE calls the destination object with the `kDETcmdDropQuery` routine selector, described next.

Your property command can use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped.

You can use the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of the object on which the item is being dropped.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

kDETcmdDropQuery

The CE calls your code resource with this routine selector when the user attempts to drop an object on the object to which your aspect template applies.

```
struct DETDropQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType;
            Boolean copyToHFS;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdDropQuery
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	modifiers	short	Modifier keys at drop time
↔	commandID	long	Command ID
↔	destinationType	AttributeType	Attribute type of new attribute
←	copyToHFS	Boolean	Copy to HFS?

DESCRIPTION

When the user drags an AOCE catalog object or HFS object and drops it onto a catalog object, the Catalogs Extension calls the code resource in the aspect of the destination object with the `kDETcmdDropQuery` routine selector. (If a destination attribute's aspect does not contain a code resource or if its code resource returns the `kDETDidNotHandle` result code, the CE calls the code resource of the aspect for the record that contains that attribute.) The `modifiers` parameter indicates which modifier keys, if any, the user was pressing when the mouse button was released.

If the user drops more than one object simultaneously on a destination, the CE calls the code resource for the destination's aspect once for each object dropped. You can call the `kDETcmdGetCommandItemN` callback routine to get information about the item being dropped.

The `commandID` and `destinationType` fields contain the CE's best guess as to the correct drop action. Possible values for the `commandID` parameter are a property number or the constants `kDETDoNothing`, `kDETMove`, `kDETDrag`, or `kDETAlias` (see page 5-171). Note that because the CE calls any code resource for the dragged object with the `kDETcmdDropMeQuery` routine selector before calling the code resource for the destination object, the values in the `commandID` and `destinationType` fields might have been provided by another code resource rather than by the CE itself.

The `destinationType` field indicates the attribute type of the attribute that the CE creates as a result of the drop if the CE copies an attribute or creates an alias to a record.

If your routine returns a result code of `kDETDidNotHandle`, the CE continues to try to determine the appropriate action. If the destination object is an attribute, the CE looks for a code resource in an aspect of the parent record.

If you wish, you can set new values for the `commandID` and `destinationType` fields and return a result code of `noErr`. The CE then uses the values you set for these parameters as input to any other code resources it finds. If no other aspect or code resource overrides these values, the CE carries out the action you specified. If the CE can't carry out the specified action, it displays a dialog box describing the problem.

You can also specify a number in the developers' property-number range (that is, `kDETFirstDevProperty` through 249) for the `commandID` parameter. The CE then sends a property command (`kDETcmdPropertyCommand`) with that number to the target aspect (that is, the code resource of the aspect that the CE indicated as the target when it called your code resource). Note that your code resource' property-command routine should treat this property number as a routine selector to determine what course of action to take. The property number you use for this purpose need not correspond to any view in the information page.

The CE combines drop operations whenever possible. Therefore, if your `kDETcmdDropQuery` routine returns the same property command for two or more dragged objects, the CE calls your code resource only once with a property command (`kDETcmdPropertyCommand`). Your property-command routine then must use the `kDETcmdGetCommandSelectionCount` and `kDETcmdGetCommandItemN` callback routines to determine which objects are being dragged and perform the appropriate action.

If you set the `copyToHFS` parameter to `true`, the CE displays a dialog box asking the user to copy the object to the desktop (that is, to create an HFS version of the object) before performing the operation. For example, if the item is a record and you set the `copyToHFS` parameter to `true`, the CE asks the user to create an information card before performing the operation. Your property-command routine can use the `kDETcmdGetCommandItemN` callback routine to get the file system specification (`FSSpec` structure) for the information card. If you set the `copyToHFS` parameter to `true`, you must set the `commandID` parameter to a property number; otherwise, the CE ignores the `copyToHFS` parameter.

CALL-FOR MASK VALUE

`kDETCallForDrops`

SEE ALSO

If the object being dragged is also a catalog object, the CE first calls the dragged object with the `kDETcmdDropMeQuery` routine selector, described on page 5-170.

Your property command can use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped.

You can use the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of each object being dropped.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

Attribute-Related Commands

The Catalogs Extension calls one of the code resource routines described in this section before the CE creates, changes, or deletes an attribute value.

kDETcmdAttributeCreation

The CE calls your code resource with this routine selector when it is about to add a new attribute value to a sublist.

```
struct DETAttributeCreationBlock {
            DETCallBlockHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdAttributeCreation`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`parent`	`PackedDSSpecPtr`	Record in which the CE creates the new attribute
→	`refNum`	`short`	Reference number for the catalog containing the record that will contain the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	`identity`	`AuthIdentity`	Authentication identity used when gaining access to the parent record
↔	`attrType`	`AttributeType`	Type of attribute being created
↔	`attrTag`	`AttributeTag`	Tag of attribute being created
↔	`value`	`Handle`	Value to write (preallocated to the size of the default attribute value, if any, or to 1 if no default; resize as needed)

DESCRIPTION

When the user clicks the Add button in an information page to add a new attribute value to a sublist, the Catalogs Extension calls the code resource for the main aspect template for attributes of that type with the `kDETcmdAttributeCreation` routine selector. The `attrType`, `attrTag`, and `value` parameters indicate the default values for the new attribute type, attribute tag, and attribute value. You can return different values for any of these parameters if you wish. Whether or not you change any of these values, return

the `kDETDidNotHandle` result code if you want the CE to create the new attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to create the new attribute value and does not create it for you. Likewise, if your routine returns an error, the CE does not create the attribute value.

After the CE calls your `kDETcmdAttributeCreation` routine and before it creates the attribute value, it calls your `kDETcmdAttributeNew` routine.

Note that the CE does not specify a target when it calls your `kDETcmdAttributeCreation` routine because the object hasn't been created yet; it always calls the code resource of the template that will be the object's main aspect template.

SPECIAL CONSIDERATIONS

You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

When the CE is about to create a new attribute value, whether in a sublist or not, it calls the `kDETcmdAttributeNew` routine, described next.

You can use the Catalog Manager's `DirAddAttributeValue` function to add an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeNew

The CE calls your code resource with this routine selector when it is about to add a new attribute value to a record.

```
struct DETAttributeNewBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdAttributeNew
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	parent	PackedDSSpecPtr	Record to which the CE adds the new attribute value
→	refNum	short	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	identity	AuthIdentity	Authentication identity used when gaining access to the parent record
↔	attrType	AttributeType	Type of attribute being created
↔	attrTag	AttributeTag	Tag of attribute being created
↔	value	Handle	Value to write (preallocated to default attribute size; resize as needed)

DESCRIPTION

When the user adds a new attribute value to a record, the Catalogs Extension calls the code resource for the parent record with the `kDETcmdAttributeNew` routine selector. The user can add a new attribute value by clicking the Add button in a template to add a new attribute value to a sublist, dragging an attribute value and dropping it onto a record, or editing a property that has not been previously edited (that is, whose value is the default value assigned by the template). The CE adds the new attribute value when the user closes the information page or when you call the `kDETcmdSaveProperty` callback routine.

The `attrType`, `attrTag`, and `value` parameters indicate the default values for the new attribute type, attribute tag, and attribute value. You can return different values for any of these parameters if you wish. Whether or not you change any of these values, return the `kDETDidNotHandle` result code if you want the CE to create the new attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to create the new attribute value and does not create it for you. Likewise, if your routine returns an error, the CE does not create the attribute value.

The target selector in the `DETTargetSpecification` structure is always `kDETSelf` when the CE calls your `kDETcmdAttributeNew` routine; the target is the aspect of the record that will contain the attribute.

When the user adds a new attribute value to a sublist, the CE calls your `kDETcmdAttributeCreation` routine before it calls your `kDETcmdAttributeNew` routine.

SPECIAL CONSIDERATIONS

If there is no input pattern for an attribute type that has an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE has no way of knowing an attribute value already exists; therefore the CE calls your `kDETcmdAttributeNew` routine every time it processes the output pattern. If your `kDETcmdAttributeNew` routine returns the `kDETDidNotHandle` result code, the record ends up containing multiple attribute values corresponding to a single set of properties. Therefore, if you do not include an input pattern for an attribute type for which you provide an output pattern, your `kDETcmdAttributeNew` routine should return the `noErr` result code when called for that attribute type.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

If the attribute value is being added to a sublist, the CE calls your `kDETcmdAttributeCreation` routine (page 5-175) before calling the `kDETcmdAttributeNew` routine.

You can use the Catalog Manager's `DirAddAttributeValue` function to add an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeChange

The CE calls your code resource with this routine selector when it is about to change an existing attribute value.

```
struct DETAttributeChangeBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            AttributeCreationID attrCID;
            Handle value;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdAttributeChange
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	parent	PackedDSSpecPtr	Record containing the attribute
→	refNum	short	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	identity	AuthIdentity	Authentication identity used when gaining access to the parent record
→	attrType	AttributeType	Type of attribute being changed
↔	attrTag	AttributeTag	Tag of attribute being changed
↔	attrCID	AttributeCreationID	CID of attribute being changed
↔	value	Handle	Value to write (preallocated to the size of the proposed attribute value; resize as needed)

DESCRIPTION

When the user changes an attribute value, the Catalogs Extension calls the code resource of the aspect that's writing the attribute with the `kDETcmdAttributeChange` routine selector. The user can change an attribute value by editing a property that has been previously edited. The CE updates the attribute value when the user closes the information page or when you call the `kDETcmdSaveProperty` callback routine.

The `attrType`, `attrTag`, and `value` parameters indicate the new values for the attribute type, attribute tag, and attribute value. You can return different values for the attribute tag and attribute value parameters if you wish. Whether or not you change either of these values, return the `kDETDidNotHandle` result code if you want the CE to change the attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to change the attribute value and does not change it for you. Likewise, if your routine returns an error, the CE does not change the attribute value.

SPECIAL CONSIDERATIONS

If there is no input pattern for an attribute type that has an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE has no way of knowing an attribute value already exists. Therefore, every time it processes the output pattern, the CE calls your `kDETcmdAttributeNew` routine rather than your `kDETcmdAttributeChange` routine.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

When the CE is about to create a new attribute value, it calls your `kDETcmdAttributeNew` routine (page 5-176).

You can use the Catalog Manager's `DirChangeAttributeValue` function to change an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeDelete

The CE calls your code resource with this routine selector when it is about to delete an existing attribute value.

```
struct DETAttributeDeleteBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr dsSpec;
            short refNum;
            AuthIdentity identity;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdAttributeDelete`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`dsSpec`	`PackedDSSpecPtr`	Catalog system specifier of the attribute value about to be deleted
→	`refNum`	`short`	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	`identity`	`AuthIdentity`	Authentication identity used when gaining access to the parent record

DESCRIPTION

When the Catalogs Extension is about to delete an attribute value, it calls the code resource of the main aspect of the attribute that's about to be deleted with the `kDETcmdAttributeDelete` routine selector. You can use the `DSSpec` structure provided in the parameter block to determine exactly which attribute value is about to be deleted.

Return the `kDETDidNotHandle` result code if you want the CE to delete the attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to delete the attribute value and does not delete it for you. Likewise, if your routine returns an error, the CE does not delete the attribute value.

The `DSSpec` structure that describes attribute values has a type of `'entn'` and the following extension value:

```
OSType               'spat'
AttributeCreationID  attributeCreationID
AttributeType        attributeName
```

The attribute creation ID uniquely identifies a specific attribute value even if there is more than one value of the same attribute type.

The `AttributeType` structure is defined as follows:

```
struct AttributeType {
   RStringHeader
   Byte body[kAttributeTypeMaxBytes];
};
```

The `attributeName` field must be packed and padded to an even number of bytes. The `AttributeType` structure is equivalent to an `RString` structure that has a length of `kAttributeTypeMaxBytes` bytes.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

You can use the Catalog Manager's `DirDeleteAttributeValue` function to delete an attribute value; see the chapter "Catalog Manager" in this book for details.

The `DSSpec`, `PackedDSSpec`,`AttributeType`, and `RString` structures are described in the chapter "AOCE Utilities" in this book. That chapter also describes utility routines that you can use to pack, unpack, and manipulate these structures.

Attribute creation IDs are returned by the `DirAddAttributeValue` function, the `DirVerifyAttributeValue` function, and the `DirFindValue` function. All of these functions are described in the chapter "Catalog Manager" in this book.

Processing Custom Lookup-Table Pattern Elements

The Catalogs Extension passes to your code resource any lookup-table attribute pattern element types that start with an uppercase letter. When the CE is processing an attribute value to set a property value and encounters a custom pattern element type, it calls your `kDETcmdPatternIn` routine. When the CE is processing a property value to create an attribute value, it calls your `kDETcmdPatternOut` routine.

kDETcmdPatternIn

The CE calls your code resource with this routine selector when it needs to use a custom lookup-table pattern element to set the value of a property.

```
struct DETPatternInBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            long dataOffset;
            short bitOffset;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdPatternIn`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`elementType`	`long`	Element type
→	`extra`	`long`	Extra field
→	`attribute`	`AttributePtr`	Attribute being parsed
↔	`dataOffset`	`long`	Offset to next byte
↔	`bitOffset`	`short`	Bit offset

DESCRIPTION

The Catalogs Extension passes to your code resource any lookup-table attribute pattern element types that start with an uppercase letter. The `property`, `elementType`, and `extra` fields contain the corresponding parts of the pattern element. The `attribute` field is a pointer to the attribute value being parsed. The `dataOffset` field is the byte offset into the attribute data of the byte to be parsed. The `bitOffset` field is the bit offset within the byte of the next bit to be parsed. The data in the attribute value is at

```
callBlockPtr->patternIn.attribute->value.bytes
```

The next bit to parse is

```
*(callBlockPtr->patternIn.attribute->value.bytes +
  callBlockPtr->patternIn.dataOffset)>>callBlockPtr->
  patternIn.bitOffset++
```

Your `kDETcmdPatternIn` routine should parse the specified attribute data starting at the byte and bit specified by the `dataOffset` and `bitOffset` fields. You can create as many properties as you wish from the attribute data. Use the `kDETcmdSetPropertyNumber`, `kDETcmdSetPropertyRString`, or `kDETcmdSetPropertyBinary` callback routines to set property values. When you are finished processing the attribute data, update the `dataOffset` and `bitOffset` fields to point to the next bit to be processed and return the `noErr` result code.

You should check the access mask of the item you are processing and set the `propertyEditable` flag accordingly.

If your routine returns the `kDETDidNotHandle` result code, the CE calls the code resource of the parent record for the attribute. If your routine returns an error, the CE stops processing attribute data.

SPECIAL CONSIDERATIONS

When the CE creates or changes attribute values, it processes only those properties that have changed and that are included in the list of properties in the lookup table. Therefore, you must use the `'prop'` pattern element in your lookup table to list each property that your code resource processes.

CALL-FOR MASK VALUE

None

SEE ALSO

Lookup-table patterns and pattern elements are described in "The Lookup-Table Resource" beginning on page 5-105.

You can use the `kDETcmdSetPropertyNumber` callback routine (page 5-227), the `kDETcmdSetPropertyRString` callback routine (page 5-228), or the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to set property values.

You can use the `kDETcmdSetPropertyEditable` callback routine (page 5-232) to set the `propertyEditable` flag.

kDETcmdPatternOut

The CE calls your code resource with this routine selector when it needs to use a custom lookup-table pattern element to write an attribute value from a property.

```
struct DETPatternOutBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            Handle data;
            long dataOffset;
            short bitOffset;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdPatternOut`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`elementType`	`long`	Element type
→	`extra`	`long`	Extra field
↔	`attribute`	`AttributePtr`	Attribute being created
↔	`data`	`handle`	Attribute data
↔	`dataOffset`	`long`	Offset to next byte to write
↔	`bitOffset`	`short`	Bit offset

DESCRIPTION

The Catalogs Extension passes to the code resource any lookup-table attribute pattern element types that start with an uppercase letter. The `property`, `elementType`, and `extra` fields contain the corresponding parts of the pattern element. The `attribute` field points to the attribute being created (the attribute value already has an attribute tag assigned, but the data length and data fields of the value have not yet been filled in). The `data` field is a handle that you can use to contain the data portion of the attribute value (and which you should resize as needed to hold the value). The `dataOffset` field is the byte offset into the attribute of the byte currently being parsed. The `bitOffset` field is the offset within the byte of the next bit to be parsed. You can change these offsets as necessary.

You can use the callback routines described in "Getting Information About Properties" beginning on page 5-213 to determine the property value. You then return the attribute data in the `data` field, resizing it as needed, and updating the `dataOffset` and `bitOffset` fields appropriately.

SPECIAL CONSIDERATIONS

Because when creating or changing attribute values, the CE processes only those properties that have changed and that are included in the list of properties in the lookup table, you must use the `'prop'` pattern element in your lookup table to list each property that your code resource processes.

CALL-FOR MASK VALUE

None

SEE ALSO

lookup-table patterns and pattern elements are described in "The Lookup-Table Resource" beginning on page 5-105.

You can use the callback routines in "Getting Information About Properties" beginning on page 5-213 to determine property values.

Synchronizing Property Values

If you derive any of your property values (including sublist items) from data outside the catalog system or from records or attributes other than the one to which your aspect applies, you can use the code resource routines described in this section to ensure that your property values are updated whenever the Catalogs Extension updates the property values that it maintains.

kDETcmdShouldSync

The CE calls your code resource with this routine selector to check whether the code resource wants to update all property values.

```
struct DETShouldSyncBlock {
            DETCallBlockTargetedHeader
            Boolean shouldSync;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdOpenSelf`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
←	`shouldSync`	`Boolean`	Should CE update data?

DESCRIPTION

The Catalogs Extension checks a catalog system flag periodically (and whenever someone calls the `kDETcmdRequestSync` callback routine) to see if the data in the catalog system has changed. If it has, the CE recalculates all the properties derived from the catalog system and updates aspects and information pages accordingly. At the time the CE checks for changes, it calls your code resource with the `kDETcmdShouldSync` routine selector. If you have derived any properties from data outside the catalog system or from records or attributes other than the one to which your aspect applies and you have reason to believe their values have changed, you should return `true` in the `shouldSync` field of the parameter block. In response, the CE updates all the properties whether data in the catalog system has changed or not, giving your code resource a chance to update all of its property values.

You can use this routine, for example, to maintain sublist items that are not directly from the catalog system.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE ignores the `shouldSync` field and behaves as if its value were `false`.

CALL-FOR MASK VALUE

```
kDETCallForSyncing
```

SEE ALSO

When the CE updates property values, it sends the `kDETcmdDoSync` routine selector, described next, to your code resource.

You can call the `kDETcmdRequestSync` callback routine (page 5-237) at any time to force the CE to check immediately whether the sublist or any properties need updating.

kDETcmdDoSync

The CE calls your code resource with this routine selector to give your code resource a chance to read in and parse its attributes.

```
struct DETDoSyncBlock {DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdOpenSelf`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

When the Catalogs Extension updates all the property values in an aspect—either because data in the catalog system has changed or because you returned `true` in the `shouldSync` field of the parameter block for the `kDETcmdShouldSync` routine—the CE calls your code resource with the `kDETcmdDoSync` routine selector. If you have derived any of your properties from outside the catalog system or from a record or attribute other than the one to which your aspect applies, you should call the `kDETcmdBreakAttribute` routine to update your sublist items and the `kDETcmdBreakAttribute` routine and the set-property routines to update your other properties.

The CE ignores the result code returned by this routine.

CALL-FOR MASK VALUE

`kDETCallForSyncing`

SEE ALSO

Call the `kDETcmdBreakAttribute` routine (page 5-224) to send to the lookup table an attribute value from outside the catalog system or from a record or attribute other than the one to which your aspect applies.

Use the set-property routines (see "Setting Value, Type, and Other Features of Properties" beginning on page 5-223) to set property values, types, and so forth for properties not derived from the catalog system or from the record or attribute to which your aspect applies

You can call the `kDETcmdRequestSync` callback routine (page 5-237) at any time to force the CE to check immediately whether the sublist or any properties need updating.

Custom Property-Type Conversions

You can assign a custom property type to a property value. When the Catalogs Extension encounters a property with a custom type that it has to use as a number or as a string, or when it encounters a number or a string that it has to store as a property with a custom type, the CE calls one of the code resource routines described in this section.

kDETcmdConvertToNumber

The CE calls your code resource with this routine selector when it encounters a property with a custom type that it has to use as a number in a view or when you call the `kDETcmdGetPropertyNumber` callback routine for a property with a custom type.

```
struct DETConvertToNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertToNumber`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
←	`theValue`	`long`	Converted value

DESCRIPTION

If you assign a custom property type to a property and then use that property in a view where a number is called for (as in a radio button, checkbox, or pop-up menu), the Catalogs Extension calls your code resource with the `kDETcmdConvertToNumber` routine selector. The CE also calls this routine if you call the `kDETcmdGetPropertyNumber` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. You must convert the property value to a number and return it in the field `theValue`.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE uses 0 as the value of the custom property. If your routine returns the `noErr` result code, the CE uses the number your routine returns in the `theValue` field.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdGetPropertyNumber` callback routine (page 5-216) to get the value of a number-type property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertToRString

The CE calls your code resource with this routine selector when it encounters a property with a custom type that it has to use as a string in a view or when you call the `GetPropertyRString` callback routine for a property with a custom type.

```
struct DETConvertToRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertToRString`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
←	`theValue`	`RStringHandle`	Handle to converted value

DESCRIPTION

If you assign a custom property type to a property and then use that property in a view that calls for a string (editable or static text), the Catalogs Extension calls your code resource with the `kDETcmdConvertToRString` routine selector. The CE also calls this routine if you call the `kDETcmdGetPropertyRString` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. You must convert the property value to an `RString` structure, allocate a handle to the `RString`, and return the handle in the field `theValue`. The CE disposes of the handle when it no longer needs it.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE uses an empty `RString` structure as the value of the property. If your routine returns the `noErr` result code, the CE uses the `RString` your routine returns in the `theValue` field.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdGetPropertyRString` callback routine (page 5-217) to get the value of a number-type property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertFromNumber

The CE calls your code resource with this routine selector when it needs to write a number to a property that has a custom property type.

```
struct DETConvertFromNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertFromNumber`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`theValue`	`UNSIGNED long`	Value to convert

DESCRIPTION

If you have assigned a custom property type to a property and then use that property in a view where the Catalogs Extension uses a number (as in a radio button, checkbox, or pop-up menu), the CE calls your code resource with the `kDETcmdConvertFromNumber` routine selector when it needs to update the property value. The CE also calls this routine if you call the `kDETcmdSetPropertyNumber` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. The `theValue` field contains the value of the property in the form of a number (unsigned long word). You must convert the property value to your custom property type and use the `kDETcmdSetPropertyBinary` callback routine

to write the result directly to the property. The CE ignores the function result of this routine.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdSetPropertyNumber` callback routine (page 5-227) to set the value of a number-type property.

You use the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to write an uninterpreted binary block to a property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertFromRString

The CE calls your code resource with this routine selector when it needs to write a string to a property that has a custom property type.

```
struct DETConvertFromRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertFromRString`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`theValue`	`RStringHandle`	Handle to value to convert

DESCRIPTION

If you have assigned a custom property type to a property and then use that property in a view where the Catalogs Extension uses a string (as in a text field), the CE calls your code resource with the `kDETcmdConvertFromRString` routine selector when it needs

to update the property value. The CE also calls this routine if you call the `kDETcmdSetPropertyRString` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. The `theValue` field contains a handle to the value of the property in the form of an `RString` structure. You must convert the property value to your custom property type and use the `kDETcmdSetPropertyBinary` callback routine to write the result directly to the property. The CE ignores the function result of this routine.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdSetPropertyRString` callback routine (page 5-228) to set the value of a string-type property.

You use the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to write an uninterpreted binary block to a property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

Custom Views and Custom Menus

You can add a custom view to a view list, and you can add custom items to the Catalogs menu. The Catalogs Extension calls the code resource routines in this section to draw custom views, handle mouse-down events in custom views, determine whether a custom menu item should be enabled, and handle the selection of your custom menu items.

kDETcmdCustomViewDraw

The CE calls your code resource with this routine selector when it needs you to draw your custom view.

```
struct DETGetCustomViewDrawBlock {
            DETCallBlockPropertyHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomViewDraw
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number

DESCRIPTION

If you include a custom view in a view list, the Catalogs Extension calls your `kDETcmdCustomViewDraw` routine when it is drawing or updating the information page that includes your custom view and when you call the `kDETcmdDirtyProperty` callback routine to indicate that the property value associated with the custom view has changed. The `property` field identifies the view to be drawn. The CE sets the graphics port to the window containing the view before calling your routine.

The CE ignores the function result for this routine.

SPECIAL CONSIDERATIONS

Your routine that draws your custom view must leave the QuickDraw state unchanged. If you change the QuickDraw state (pen pattern, background color, and so forth) while drawing your custom view, you must restore it to its original values before returning.

CALL-FOR MASK VALUE

None

SEE ALSO

View lists are described in "View Lists" beginning on page 5-123.

The `kDETcmdDirtyProperty` callback routine is described on page 5-233.

kDETcmdCustomViewMouseDown

The CE calls your code resource with this routine selector when a mouse-down event occurs in your custom view.

```
struct DETCustomViewMouseDownBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomViewMouseDown
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	theEvent	EventRecord	The event record for the mouse-down

DESCRIPTION

If you include a custom view in a view list, the Catalogs Extension calls your `kDETcmdCustomViewMouseDown` routine when a mouse-down event occurs in your custom view. The mouse coordinates in the event record are global; you can use the `GlobalToLocal` QuickDraw routine to obtain the local coordinates.

The CE sets the graphics port to the window containing the view before calling your routine.

The property field identifies the view in which the event occurred.

If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr` or `kDETDidNotHandle`, the CE does no further processing of the event.

CALL-FOR MASK VALUE

None

SEE ALSO

View lists are described in "View Lists" beginning on page 5-123.

kDETcmdCustomMenuEnabled

The CE calls your code resource with this routine selector to determine whether to enable a menu item that you have added to the Catalogs menu.

```
struct DETCustomMenuEnabledBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
            Boolean enable;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdCustomMenuSelected`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`menuTableParameter`	`short`	`property` field from custom menu table
←	`enable`	`Boolean`	Enable the menu item?

DESCRIPTION

If you add a custom menu item to the Catalogs menu by including a `kDETInfoPageMenuEntries` resource in your information page template, the Catalogs Extension calls your `kDETcmdCustomMenuEnabled` routine when the user opens the Catalogs menu. To enable the menu item identified by the `menuTableParameter` field, return `noErr` with the `enable` parameter set to `true` or return `kDETDidNotHandle`. To disable the menu item, return `noErr` with the `enable` parameter set to `false` or return an error.

CALL-FOR MASK VALUE

None

SEE ALSO

If you enable the menu item and the user chooses it, the CE calls your code resource with the `kDETcmdCustomMenuSelected` routine selector, described next.

The `kDETInfoPageMenuEntries` resource is described on page 5-137.

kDETcmdCustomMenuSelected

The CE calls your code resource with this routine selector when the user chooses a menu item that you have added to the Catalogs menu.

```
struct DETCustomMenuSelectedBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdCustomMenuSelected`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`menuTableParameter`	`short`	`property` field from custom menu table

DESCRIPTION

If you add a custom menu item to the Catalogs menu by including a `kDETInfoPageMenuEntries` resource in your information page template, the Catalogs Extension calls your `kDETcmdCustomMenuSelected` routine when the user chooses your menu item. The `menuTableParameter` field contains the menu parameter from the `kDETInfoPageMenuEntries` resource for the item the user chose.

If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr` or `kDETDidNotHandle`, the CE does no further processing of the menu selection.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETInfoPageMenuEntries` resource is described on page 5-137.

The CE calls your `kDETcmdCustomMenuEnabled` routine (page 5-194) to determine whether to enable your menu item.

CE-Provided Functions That Your Code Resource Can Call

Your code resource can call certain routines within the AOCE Catalogs Extension to perform such functions as changing the call-for mask, returning information from the CE, or changing the value of a variable maintained by the CE.

Note
Because the CE first calls your code resource, and then your code calls the CE back to perform these functions, these routines are referred to here as "callback routines." ◆

Calling CE-Provided Functions

The parameter block that the Catalogs Extension passes to your code resource includes the address of the entry point for CE callback routines. To execute one of these routines, you can use the `CallBackDET` macro, described in this section.

▲ **WARNING**
Because the parameter block passed by the CE to your code resource contains data that is private to the CE in addition to data intended for your code resource, it is essential that you pass back to the CE the same parameter block that it passed you without altering any of the reserved fields. You cannot reuse a parameter block you saved from a previous time that the CE called your code resource; doing so can cause the Finder to crash. ▲

IMPORTANT
When you call a template callback routine, you must make sure that the system is in the same state as it was in when the CE called your code resource. Changing such things as the current resource or the heap zone will cause the callback routine to fail. ▲

CallBackDET

The `CallBackDET` macro calls any AOCE template callback routine.

```
CallBackDET(callBlockPtr, callBackBlockPtr);
```

`callBlockPtr`
A pointer to the parameter block that the CE passed to your code resource.

`callBackBlockPtr`
A pointer to the parameter block that you are providing to the CE callback routine.

The `CallBackDET` macro passes the parameter blocks you provide to the Catalogs Extension callback routine entry point. The function gets the address for this entry point from the `callBack` field of the AOCE template call block that the CE passes to your code resource.

ASSEMBLY-LANGUAGE INFORMATION

The `CallBackDET` macro is implemented entirely in the interface file. There is no trap that corresponds to this macro.

Testing Your Code Resource

The Catalogs Extension provides the `kDETcmdBeep` callback routine so that you can make sure your code resource is being called correctly and is making callbacks correctly.

kDETcmdBeep

This callback routine calls the toolbox `SysBeep` routine.

```
struct DETBeepBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdBeep
```

DESCRIPTION

You can use the `kDETcmdBeep` callback routine to test that your code resource and its callback routines are working as you expect.

RESULT CODES

`noErr`	0	No error

Changing the Call-For Mask

You can use the `kDETcmdChangeCallFors` callback routine to modify the call-for mask and therefore change the list of events that result in calls to your code resource.

kDETcmdChangeCallFors

This callback routine changes the call-for mask to a new value.

```
struct DETChangeCallForsBlock {
            DETCallBackBlockTargetedHeader;
            long newCallFors;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdChangeCallFors
→	target	DETTargetSpecification	Target specifier
→	newCallFors	long	New call-for mask

DESCRIPTION

You can modify the call-for mask for the code resource associated with your aspect template at any time. You use the `target` parameter to specify the aspect template whose code resource is your target and the `newCallFors` parameter to provide a new call-for mask. The Catalogs Extension uses the same the call-for mask for every aspect created from the aspect template.

Most code resources set the call-for mask at template initialization time and never change it. However, you might want to change the call-for mask before you call a callback that might result in additional unwanted calls to your code resource. In that case you can set the call-fors mask to `kDETCallForNothing`, call the callback routine, and then reset the call-for mask to its former value.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The target specifier is described in "Target Specifier" on page 5-142.

The call-for mask is described in "Call-For Mask" on page 5-149.

Process Control

The routines in this section give you some control over process switching on the user's computer. You should use the `kDETcmdAboutToTalk` callback routine when you want to display a dialog box or otherwise interact with the user outside of an information page. You can use the `kDETcmdBusy` routine to initiate a process switch to allow some other process to complete before returning control to you.

kDETcmdAboutToTalk

This callback routine brings the Finder to the front and disables the watch cursor.

```
struct DETAboutToTalkBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdAboutToTalk
```

DESCRIPTION

You should call this routine whenever you are about to display a dialog box or interact with the user in any way outside of the information page. When your code resource returns control to the CE, the CE terminates this state.

RESULT CODES

`noErr`	0	No error

kDETcmdBusy

This callback routine initiates a process switch and prevents user action.

```
struct DETBusyBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdBusy
```

DESCRIPTION

The `kDETcmdBusy` callback routine initiates a process switch; its effect is similar to that of the `WaitNextEvent` function. It sounds a system beep if the user presses the mouse button or a key. You can use this routine to give other processes some time to complete an operation.

In general, code resource routines should complete operation quickly and return. You should use this routine only if you have a special need to cause a process switch before returning control to the Finder.

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `WaitNextEvent` function is described in the "Event Manager" chapter of *Inside Macintosh: Macintosh Toolbox Essentials*.

Handling Drags and Drops

When the user drags one or more objects and drops them onto another object, the Catalogs Extension calls your code resource if either object was an AOCE catalog object for which you provided an aspect template. Your code resource can use the routines in this section to determine the number of objects dropped and the natures of the objects involved.

kDETcmdGetCommandSelectionCount

This callback routine returns the command selection count.

```
struct DETGetCommandSelectionCountBlock {
            DETCallBackBlockHeader
            long count;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetCommandSelectionCount`
←	`count`	`long`	Command selection count

DESCRIPTION

When the user drops one or more objects onto a catalog object for which you have provided an aspect template, the Catalogs Extension calls your code resource (if any) with the `kDETcmdDropQuery` routine selector. Your code resource returns a value telling the CE how to handle the drop. One possible value you can return is a property number, which causes the CE to call your code resource with a property command. When the CE calls your code resource with the `kDETcmdPropertyCommand` routine selector resulting from a drop, you can call the `kDETcmdGetCommandSelectionCount` callback routine to find out how many objects are being dropped. Then for each item, call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object being dropped.

The count of objects begins with 1; that is, if one object is being dropped, the `count` field contains a 1.

RESULT CODES

noErr	0	No error

SEE ALSO

Call the kDETcmdGetCommandItemN callback routine (described next) to determine the nature of the object being dropped.

The kDETcmdDropQuery routine is described on page 5-172.

The kDETcmdPropertyCommand routine is described on page 5-159.

kDETcmdGetCommandItemN

This callback routine returns a specific command selection item.

```
struct DETGetCommandItemNBlock {
      DETCallBackBlockHeader
      long itemNumber;
      DETItemType itemType;
      union {
         DETFSInfo** fsInfo;
         struct {
                  PackedDSSpecPtr* dsSpec;
                  short refNum;
                  AuthIdentity identity;
         } ds;
         PackedDSSpecPtr* dsSpec;
         LetterSpec** ltrSpec;
      } item;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCommandItemN
→	itemNumber	long	Number of item to retrieve, starting at 1
←	itemType	DETItemType	Type of item to be returned
←	item	union	Address or letter specifier of item

DESCRIPTION

When the user drops one or more objects onto a catalog object for which you have provided an aspect template, the Catalogs Extension calls your code resource (if any) with the kDETcmdDropQuery routine selector once for each object dropped. Your

drop-query routine can call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object being dropped. When the user drags a catalog object and drops it onto another catalog object or onto an HFS object, the CE calls the code resource in the aspect of the dragged object with the `kDETcmdDropMeQuery` routine selector. Your drop-me query routine can call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the destination object. Both your drop-query and drop-me query routines return a value telling the CE how to handle the drop.

One possible value you can return is a property number, which causes the CE to call your code resource with a property command. The CE groups all property commands that use the same property number resulting from a drop and calls your code resource once. When the CE calls your code resource with the `kDETcmdPropertyCommand` routine selector resulting from a drop, you can call the `kDETcmdGetCommandSelectionCount` callback routine to find out how many items are being dropped. Then for each item, call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object being dropped.

The `kDETcmdGetCommandItemN` callback routine returns information about the item you specify in the format you specify with the `itemType` parameter. The possible values of the `itemType` parameter are as follows:

```
enum DETItemType {
   kDETHFSType = 0,              /* HFS item type */
   kDETDSType,                   /* catalog service item type */
   kDETMailType,                 /* mail (letter) item type */
   kDETMoverType,                /* sounds, fonts, etc., from inside
                                    a suitcase or system file */
   kDETLastItemType = 0xF0000000 /* force itemType to be a long */
};

typedef enum DETItemType DETItemType;
```

The `item` parameter is a union of several structures, as shown in the `DETGetCommandItemNBlock` structure at the beginning of this routine description.

If you request an HFS item type, the routine returns a handle to a file system information structure. This structure includes the file system specification structure for the HFS object, plus its file type, file creator, and Finder flags. The file system information structure is defined by the `DETFSInfo` data type.

```
struct DETFSInfo {
   OSType fileType;        /* file type */
   OSType fileCreator;     /* file creator */
   unsigned short fdFlags; /* Finder flags */
   FSSpec fsSpec;          /* FSSpec */
};

typedef struct DETFSInfo DETFSInfo;
```

If you request a catalog service item type, the routine returns a `ds` structure.

```
struct {
        PackedDSSpecPtr* dsSpec;   /* DSSpec for item */
        short refNum;              /* refnum for returned address */
        AuthIdentity identity;     /* identity for returned address */
     } ds;
```

This structure includes a handle to a catalog service specification structure (`DSSpec`) that identifies the item, a personal catalog reference number if the item is in a personal catalog, and an authentication identity if the item is in a catalog other than a personal catalog. The CE allocates the handle to the `DSSpec` structure but you must dispose of the handle when you are finished with it.

If you request a mail item type, the routine returns a handle to a letter-specification (`ltrSpec`) structure. The CE allocates the handle but you must dispose of the handle when you are finished with it. You can use the letter-specification structure in the `SMPGetLetterInfo` function to get information about the letter. The letter-specification structure is defined by the `LetterSpec` data type.

```
struct LetterSpec {
   unsigned long  spec[3];
};
```

Your `kDETcmdDropQuery` or `kDETcmdDropMeQuery` routine might receive a `kDETMoverType` item type, indicating a Finder object, such as a font or sound, that is inside a suitcase or system file. To manipulate such objects, you must set the `copyToHFS` parameter to `true` in your `kDETcmdDropQuery` or `kDETcmdDropMeQuery` routine so that the user will copy them to HFS objects and try the drop again.

If the object for which you request information is not available in the format you request, the routine returns the `kDETRequestedTypeUnavailable` result code.

It is generally best to request item types in the order you prefer to deal with them. For example, if you want to do something with a catalog object, you might ask first for an item of type `kDETDSType`. If there are no such objects and your code resource can handle HFS objects (such as information cards), you might next try the `kDETHFSType` item type.

RESULT CODES

`noErr`	0	No error
`kDETInvalidCommandItemNumber`	–15007	Command item number out of range
`kDETUnableToGetCommadnItemSpec`	–15008	Unable to retrieve information about item (possibly out of memory)
`kDETRequestedTypeUnavailable`	–15009	Item could not be represented in the specified format

SEE ALSO

The `kDETcmdDropQuery` routine is described on page 5-172.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

You can use the letter-specification structure in the `SMPGetLetterInfo` function to get information about the letter; the `SMPGetLetterInfo` function is described in the chapter "Standard Mail Package" in this book.

Working With Templates

The routines in this section allow your code resource to determine how many templates have been loaded by the system, to locate template files and template resources, and to close and unload all templates.

kDETcmdTemplateCounts

This callback routine returns the numbers of aspect and information page templates in the system.

```
struct DETTemplateCounts {
            DETCallBackBlockHeader
            long aspectTemplateCount;
            long infoPageTemplateCount;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdTemplateCounts`
←	`aspectTemplateCount`	`long`	Number of aspect templates
←	`infoPageTemplateCount`	`long`	Number of information page templates

DESCRIPTION

You can use the information returned by the `DETcmdTemplateCounts` callback routine if you want to iterate through all of the templates in the system; for example, to search for a custom resource of a specific type.

RESULT CODES

`noErr`	0	No error

kDETcmdGetTemplateFSSpec

This callback routine returns the file system specification for a template file.

```
struct DETGetTemplateFSSpecBlock {
            DETCallBackBlockTargetedHeader
            FSSpec fsSpec;
            short baseID;
            long aspectTemplateNumber;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetTemplateFSSpec`
→	`target`	`DETTargetSpecification`	Target specifier
←	`fsSpec`	`FSSpec`	`FSSpec` of file containing the template
←	`baseID`	`short`	Base resource ID of this template
←	`aspectTemplateNumber`	`long`	The template number for this aspect template

DESCRIPTION

The `kDETcmdGetTemplateFSSpec` callback routine returns an `FSSpec` structure for the file containing the target template.

You can use a template index number (the number that the Catalogs Extension assigned to this aspect template when it loaded the template into memory) for the target specifier in the `target` field. Whatever type of target selector you use, this function returns the index number of the template.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The `FSSpec` structure is described in the chapter "Introduction to File Management" in *Inside Macintosh: Files*.

Target selectors are described in "Target Specifier" on page 5-142.

kDETcmdGetResource

This callback routine returns a template resource.

```
struct DETGetResourceBlock {
            DETCallBackBlockPropertyHeader
            ResType resourceType;
            Handle theResource;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetResource`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`resourceType`	`ResType`	Resource type
←	`theResource`	`Handle`	Handle to the resource

DESCRIPTION

The `kDETcmdGetResource` callback routine returns a handle to a resource. This resource has a resource ID equal to the property number plus the template's base ID and has the resource type you specify. If the call-for mask is set appropriately, the routine calls your code resource with the `kDETcmdDynamicResource` routine selector for all resources except those listed in a forwarder template. It takes the record type resource (at offset `kDETRecordType`), attribute type resource (`kDETAttributeType`), and attribute value tag resource (`kDETAttributeValueTag`) from the version of the template that's stored in memory, because these resources might have been added by a forwarder template or by the `kDETcmdDynamicForwarders` code-resource routine.

You can use the `kDETAspectTemplate` and `kDETInfoPageTemplate` target selectors in the target specifier you use with this callback routine. These target selectors allow you to specify the index number of the template assigned by the Catalogs Extension when it loads the template into memory. You might want to use these target selectors, for example, to search for every template in memory that contains a resource of a specific type.

If the targeted template does not contain the specified resource, the routine returns the `resNotFound` result code.

You must dispose of the resource handle when you have finished using it.

RESULT CODES

`noErr`	0	No error
`resNotFound`	–192	Could not find specified resource
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The `kDETcmdDynamicResource` routine is described on page 5-156.

Target selectors are described in "Target Specifier" on page 5-142.

Forwarder templates are described in "Components of Forwarder Templates" on page 5-138, and the `kDETcmdDynamicForwarders` code-resource routine is described on page 5-155.

kDETcmdUnloadTemplates

This callback routine unloads all templates from memory.

```
struct DETUnloadTemplatesBlock {
   DETCallBackBlockHeader
};
```

Parameter block

→ `reqFunction` `DETCallBackFunctions` `kDETcmdUnloadTemplates`

DESCRIPTION

This callback routine causes the Catalogs Extension to close all template-related windows, release all memory used by templates, and delete all templates and template-related data structures from memory. At that point, you can put templates and new versions of templates in the Extensions folder. The CE loads the new templates the next time they are needed.

This routine should not normally be called by a template. It is provided for the convenience of template developers so that you don't need to reboot your test system every time you want to try a new version of a template or add a new template to the system.

RESULT CODES

`noErr`	0	No error

Working With Catalog Objects

The routines in this section return a catalog system specification for an object and let your code resource open a catalog object.

kDETcmdGetDSSpec

This callback routine returns a catalog system specification structure for the targeted object.

```
struct DETGetDSSpecBlock {
            DETCallBackBlockTargetedHeader
            PackedDSSpecPtr* dsSpec;
            short refNum;
            AuthIdentity identity;
            Boolean isAlias;
            Boolean isRecordRef;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetDSSpec`
→	`target`	`DETTargetSpecification`	Target specifier
←	`dsSpec`	`PackedDSSpecPtr*`	Handle to `DSSpec`
←	`refNum`	`short`	Reference number for the catalog containing the object (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
←	`identity`	`AuthIdentity`	Authentication identity used to gain access to the catalog containing the object
←	`isAlias`	`Boolean`	True if this `DSSpec` is for an alias to a record
←	`isRecordRef`	`Boolean`	Reserved

DESCRIPTION

The Catalogs Extension allocates the handle to store the `PackedDSSpec` structure returned by this function. Your code resource must deallocate the handle when done.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The `PackedDSSpec` structure and functions that you can use to unpack it are described in the chapter "AOCE Utilities" in this book.

You can use the `kDETcmdOpenDSSpec` callback routine (described next) to open the object described by the `DSSpec` structure.

kDETcmdOpenDSSpec

This callback routine opens the object for which you supply a catalog specification (`DSSpec`) structure.

```
struct DETOpenDSSpecBlock {
            DETCallBackBlockHeader
            PackedDSSpecPtr dsSpec;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdOpenDSSpec`
→	`dsSpec`	`PackedDSSpecPtr`	`DSSpec` of object to be opened

DESCRIPTION

You can use the `kDETcmdOpenDSSpec` callback routine to open any object for which you have a catalog specification structure (`DSSpec`). The exact effect of opening the object depends on the object; the Catalogs Extension might open an information page, or the object's code resource might perform some other action. The CE does not actually open the object until after your code resource returns.

RESULT CODES

`noErr`	0	No error
`kDETInvalidDSSpec`	–15010	Could not resolve `DSSpec`

SEE ALSO

You can use the preceding routine, `kDETcmdGetDSSpec`, to obtain the `DSSpec` structure for a catalog object.

Edit-Text Routines

The callback routines in this section give your code resource some control over edit-text views. The first routine, `kDETcmdGetOpenEdit`, returns the property number of an edit-text view. The `kDETcmdCloseEdit` routine closes a specific edit-text view.

kDETcmdGetOpenEdit

This callback routine returns the property number of the edit-text view that the user is currently editing.

```
struct DETGetOpenEditBlock {
            DETCallBackBlockTargetedHeader
            short viewProperty;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetOpenEdit`
→	`target`	`DETTargetSpecification`	Target specifier
←	`viewProperty`	`short`	The property number of the view being edited

DESCRIPTION

If no edit-text view is currently being edited, this function returns the value `kDETNoProperty` in the `viewProperty` field.

Note that, because this routine can be targeted, you can use it in the code resource for a parent to get information about the information page of a child.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETNoSuchView`	–15013	No view found with specified property number

kDETcmdCloseEdit

This callback routine closes the currently open edit-text view.

```
struct DETCloseEditBlock {
            DETCallBackBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdCloseEdit`
→	`target`	`DETTargetSpecification`	Target specifier

DESCRIPTION

This callback routine removes the focus box (if any) from the currently open edit-text view, removes the insertion point from the view, and finalizes the edit. After you call this routine, the user must click again within the view to reopen the edit text. The information page must be open when you call this routine; if it is not, the function returns the `kDETInfoPageNotOpen` result code.

Note that, because this routine can be targeted, you can use it in the code resource for a parent to affect the information page of a child.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETInfoPageNotOpen`	–15012	Information page not open

SEE ALSO

To determine the property number of the currently open edit-text view, use the `kDETcmdGetOpenEdit` callback routine (page 5-211).

Getting Information About Properties

The routines described in this section provide information about properties. Note that because the Catalogs Extension looks up information in catalogs asynchronously, it might not have found the information you are asking for if it has not had time to complete its search. You can use the value of the property `kDETPastFirstLookup` to determine whether the CE has completed its catalog search. This property value equals 0 until the search is complete, after which it equals 1.

When your code resource requests the value of a property, you use the `kDETcmdGetPropertyNumber`, `kDETcmdGetPropertyRString`, or `kDETcmdGetPropertyBinary` callback routine to obtain the property as a specific type. If the actual property containing the value is of a different type, the CE automatically converts the value to the requested type, calling the template code resource if the source property is a custom type.

Table 5-15 summarizes the CE's actions when you request a property value. See Table 5-16 on page 5-223 for the conversions the CE performs when you set a property value.

Table 5-15 Property-type conversions on requesting a property value

Callback routine	Property type	Conversion
`kDETcmdGetPropertyNumber`	Number	None
`kDETcmdGetPropertyNumber`	String	Interprets as number, ignoring non-numeric characters
`kDETcmdGetPropertyNumber`	Binary	Takes first 4 bytes of binary data
`kDETcmdGetPropertyNumber`	Custom	Calls code resource `kDETcmdConvertToNumber` routine
`kDETcmdGetPropertyRString`	Number	Converts unsigned number to string
`kDETcmdGetPropertyRString`	String	None
`kDETcmdGetPropertyRString`	Binary	Interprets binary data as an `RString` structure
`kDETcmdGetPropertyRString`	Custom	Calls code resource `kDETcmdConvertToRString` routine
`kDETcmdGetPropertyBinary`	Number	None
`kDETcmdGetPropertyBinary`	String	None
`kDETcmdGetPropertyBinary`	Binary	None
`kDETcmdGetPropertyBinary`	Custom	None

The `kDETcmdGetPropertyChanged` and `kDETcmdGetPropertyEditable` routines get the values of the property-changed and property-editable flags for a specific property.

kDETcmdGetPropertyType

This callback routine returns the type of the specified property.

```
struct DETGetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short propertyType;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyType
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyType	short	Property type

DESCRIPTION

Standard property types are `kDETPrTypeNumber` for numbers, `kDETPrTypeString` for strings, or `kDETPrTypeBinary` for binary blocks. You can also define your own property types. If you have never explicitly set the type of a property—either by using a lookup-table pattern element, by using a resource type for the property that confers a default property type (`'rstr'`, `'detn'`, or `'detb'`), or by using the `kDETcmdSetPropertyType` callback routine—then the property is of type `kDETPrTypeBinary`.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

Property types are described in "Properties" beginning on page 5-84.

Code-resource routines that you can provide to convert custom property types to and from standard property types are described in "Custom Property-Type Conversions" beginning on page 5-188.

You can use lookup-table elements to set property types. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

You can use the `kDETcmdSetPropertyType` callback routine (page 5-225) to change the type of a property.

kDETcmdGetPropertyNumber

This callback routine returns the value of a property as a number.

```
struct DETGetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long propertyValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyNumber`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyValue`	`long`	Property value

DESCRIPTION

A property of type `kDETPrTypeNumber` is stored internally as an unsigned long word, and this function returns that value.

If the property is of type `kDETPrTypeString`, the `kDETcmdGetPropertyNumber` function removes all nonnumeric characters and returns the remaining string as a number. The function does not recognize minus signs (–), hexadecimal signs ($ or 0x), or other special symbols when converting strings to numbers.

If the property is of type `kDETPrTypeBinary`, the function returns the first 4 bytes of the property value as a number.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property before you get its value.

To get a property value as a string, use the `kDETcmdGetPropertyRString` callback routine, described next.

To get a property value as a binary block, use the `kDETcmdGetPropertyBinary` callback routine (page 5-219).

kDETcmdGetPropertyRString

This callback routine returns the value of a property as an `RString` structure.

```
struct DETGetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringHandle propertyValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyRString`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyValue`	`RStringHandle`	Handle to property value

DESCRIPTION

A property of type `kDETPrTypeString` is stored internally as an `RString` structure, and this callback routine returns that value. If the property is of type `kDETPrTypeNumber`, this routine converts the number to an `RString`. If the property is of type `kDETPrTypeBinary`, this routine assumes the binary block contains an `RString` and returns it as such.

When this callback routine completes with the `noErr` result code, the Catalogs Extension allocates the handle in the `propertyValue` field. It is your responsibility to deallocate it when done. The function always returns a valid handle, even if the string is of length 0.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the kDETcmdGetPropertyType callback routine (page 5-214) to determine the type of a property before you get its value.

To get a property value as a number, use the kDETcmdGetPropertyNumber callback routine (page 5-216).

To get a property value as a binary block, use the kDETcmdGetPropertyBinarySize callback routine, described next.

kDETcmdGetPropertyBinarySize

This callback routine returns the size of a property value.

```
struct DETGetPropertyBinarySizeBlock {
            DETCallBackBlockPropertyHeader
            long propertyBinarySize;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyBinarySize
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyBinarySize	long	Property size

DESCRIPTION

This function treats the property as a binary block regardless of the property type, returning the number of bytes in the property value. You can use this function to determine how many bytes of data will be returned by the `kDETcmdGetPropertyBinary` function.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

This function tells you how many bytes of data will be returned by the `kDETcmdGetPropertyBinary` function (described next) for a given property.

kDETcmdGetPropertyBinary

This callback routine returns the value of a property as a binary block.

```
struct DETGetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Handle propertyValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyBinary`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyValue`	`Handle`	Handle to property value

DESCRIPTION

The `kDETcmdGetPropertyBinary` function returns the value of a property as an uninterpreted binary block, regardless of the type of the property. If the property is of type `kDETPrTypeString`, for example, this function returns the `RString` character set and data length fields along with the string itself as binary data.

When this callback routine completes with the `noErr` result code, the Catalogs Extension allocates the handle in the `propertyValue` field. It is your responsibility to deallocate the handle when done.

SPECIAL CONSIDERATIONS

The size of the handle returned by this routine is not the size of the property value. Use the `kDETcmdGetPropertyBinarySize` callback routine to determine the size of a property value.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property before you get its value.

You can use the `kDETcmdGetPropertyBinarySize` callback routine (page 5-218) to determine the size of a property value before calling the `kDETcmdGetPropertyBinary` function.

To get a property value as a number, use the `kDETcmdGetPropertyNumber` callback routine (page 5-216).

To get a property value as a string, use the `kDETcmdGetPropertyRString` callback routine (page 5-217).

kDETcmdGetPropertyChanged

This callback routine indicates whether a property value has been changed.

```
struct DETGetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyChanged`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyChanged`	`Boolean`	Is property-changed flag set?

DESCRIPTION

This function returns the value of the property-changed flag, which indicates whether the user has changed this property.

If the property-changed flag for this property is set, the Catalogs Extension saves the value of the property when the user closes the information page. You can check the value of this field and save the property value yourself if you have a special need to do so. In addition, if other portions of your display depend on the value of this property, you can use this knowledge to update the display.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdSetPropertyChanged` callback routine (page 5-231) to set the property-changed flag for a property.

kDETcmdGetPropertyEditable

This callback routine indicates whether a property can be edited by the user or whether a control view is enabled.

```
struct DETGetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyEditable
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyEditable	Boolean	Is property editable?

DESCRIPTION

The access controls for the dNode, record, and attribute determine whether a property is editable. You can also use the `kDETcmdSetPropertyEditable` callback routine to make a text view uneditable or to disable a control view. Note that if a property is not editable, neither is a text view based on that property. Also, controls that would change the value of that property are not enabled.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdSetPropertyEditable` callback routine (page 5-232) to set or clear the property-editable flag.

Setting Value, Type, and Other Features of Properties

The routines in this section let your code resource set property values and other property features. The first routine, `kDETcmdBreakAttribute`, sends an attribute to the lookup table to create or update one or more properties. The `kDETcmdSetPropertyType` routine sets the type of a property. The `kDETcmdSetPropertyNumber`, `kDETcmdSetPropertyRString`, and `kDETcmdSetPropertyBinary` commands set the values of properties, converting the types of the values as shown in Table 5-16. See Table 5-15 on page 5-214 for the conversions the Catalogs Extension performs when you get a property value.

Table 5-16 Property-type conversions on setting a property value

Callback routine	Property type	Conversion
`kDETcmdSetPropertyNumber`	Number	None
`kDETcmdSetPropertyNumber`	String	Converts unsigned number to string
`kDETcmdSetPropertyNumber`	Binary	Sets type to number, then sets value
`kDETcmdSetPropertyNumber`	Custom	Calls code resource `kDETcmdConvertFromNumber` routine
`kDETcmdSetPropertyRString`	Number	Interprets as number, ignoring non-numeric characters
`kDETcmdSetPropertyRString`	String	None
`kDETcmdSetPropertyRString`	Binary	Sets type to string, then sets value
`kDETcmdSetPropertyRString`	Custom	Calls code resource `kDETcmdConvertFromRString` routine
`kDETcmdSetPropertyBinary`	Number	Sets value, leaving type as number
`kDETcmdSetPropertyBinary`	String	Sets value, leaving type as string
`kDETcmdSetPropertyBinary`	Binary	None
`kDETcmdSetPropertyBinary`	Custom	Sets value, leaving custom type as defined by developer

The `kDETcmdSetPropertyChanged` and `kDETcmdSetPropertyEditable` routines set the property-changed and property-editable flags for a specific property. The `kDETcmdDirtyProperty` routine causes the CE to redraw the view associated with a property. The `kDETcmdSaveProperty` causes the CE to save a property immediately.

kDETcmdBreakAttribute

This callback routine causes the CE to parse an attribute.

```
struct DETBreakAttributeBlock {
            DETCallBackBlockTargetedHeader
            AttributePtr breakAttribute;
            Boolean isChangeable;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdBreakAttribute`
→	`target`	`DETTargetSpecification`	Target specifier
→	`breakAttribute`	`AttributePtr`	Attribute to parse
→	`isChangeable`	`Boolean`	Can user change value?

DESCRIPTION

The Catalogs Extension uses the lookup table of the target aspect to process the attribute pointed to by the `breakAttribute` field. This routine allows you to use an attribute value from a different record or from outside the catalog system. The `isChangeable` field indicates whether the user can edit the value so that the CE can set the property-editable flag for the property.

Note
A lookup table can contain only one input pattern and one output pattern for each attribute type. Therefore, although the CE places no restriction on the number of attribute values that can be assigned to each attribute type, lookup-table patterns are designed to work only for those multivalued attributes that appear in sublists. ◆

SPECIAL CONSIDERATIONS

If your `kDETcmdDoSync` code resource routine uses the `kDETcmdBreakAttribute` callback to supply sublist items from outside the AOCE catalog system, you must supply a unique type and CID to each item, and you must use the same type and CID for that item every subsequent time the CE calls your `kDETcmdDoSync` routine. Otherwise, the CE deletes the item as obsolete.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

You must call the `kDETcmdBreakAttribute` callback routine from your `kDETcmdDoSync` code resource routine (page 5-186) if you are providing attribute values from outside the AOCE catalog system.

Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

kDETcmdSetPropertyType

This callback routine sets a property's type.

```
struct DETSetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short newType;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyType`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`newType`	`short`	New property type

DESCRIPTION

You can use the `kDETcmdSetPropertyType` callback routine to set the type of a property. The standard AOCE property types are `kDETPrTypeNumber` for numbers, `kDETPrTypeString` for strings, or `kDETPrTypeBinary` for binary blocks. You can also define your own property types. Because Apple Computer, Inc., reserves all property-type values less than or equal to 0, you must give your property type a positive value.

Note that this routine just sets the property's type; it does not convert the property value to the new type. You should convert the property value or redraw the display as appropriate.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

Property types are described in "Properties" beginning on page 5-84.

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property.

You can use the `kDETcmdDirtyProperty` callback routine (page 5-233) to cause the CE to redraw the view.

Whenever the CE needs to convert to or from one of your private property types, it calls your code resource. Code resource routines that you can provide to convert custom property types to and from standard property types are described in "Custom Property-Type Conversions" beginning on page 5-188.

You can use lookup-table elements to set property types. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

kDETcmdSetPropertyNumber

This callback routine sets the value of a property using a number as input.

```
struct DETSetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long newValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyNumber`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`newValue`	`long`	New property value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected views to be redrawn. If the property is of type `kDETPrTypeString`, the Catalogs Extension converts the unsigned number in the `newValue` field to an `RString`. If the property is of type `kDETPrTypeBinary`, the CE sets the property type to `kDETPrTypeNumber` before setting its value. If the property is a custom type, the CE calls the code resource's `kDETcmdConvertFromNumber` routine to convert the value and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyNumber` callback routine (page 5-216) to determine the value of a number property.

To cause the CE to save the new property value, call the `kDETcmdSetPropertyChanged` callback routine (page 5-231).

kDETcmdSetPropertyRString

This callback routine sets the value of a property using an `RString` as input.

```
struct DETSetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringPtr newValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyRString`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`newValue`	`RStringPtr`	Pointer to new property value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected views to be redrawn. If the property is of type `kDETPrTypeNumber`, the Catalogs Extension removes all nonnumeric characters and uses the remaining number to set the property value. The function does not recognize minus signs (–), hexadecimal signs ($ or 0x), or other special symbols when converting strings to numbers. If the property is of type `kDETPrTypeBinary`, the CE sets the property type to `kDETPrTypeString` before setting its value. If the property is a custom type, the CE calls the code resource's `kDETcmdConvertFromRString` routine to convert the value and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyRString` callback routine (page 5-217) to determine the value of a string property.

To cause the CE to save the new property value, call the `kDETcmdSetPropertyChanged` callback routine (page 5-231).

kDETcmdSetPropertyBinary

This callback routine sets the value of a property using a binary value as input.

```
struct DETSetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Ptr newValue;
            long newValueSize;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyBinary`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`newValue`	`Ptr`	Pointer to new property value
→	`newValueSize`	`long`	Size of new value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected view to be redrawn. If the property is of type `kDETprTypeNumber`, the Catalogs Extension assumes the binary value is a number and sets the property length accordingly. If the property is of type `kDETprTypeString`, the CE uses the `newValueSize` parameter as the length of the property but and sets the property value to the binary block you provide. (Note that the CE will subsequently assume this property value to be an `RString` structure, interpreting the first 4 bytes as the `charSet` and `dataLength` fields.) If the property is a custom type, the CE sets the property length to the size in the `newValueSize` parameter and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyBinary` callback routine (page 5-219) to determine the value of a binary property.

To cause the CE to save the new property value, call the `kDETcmdSetPropertyChanged` callback routine (described next).

The `RString` data structure is defined in the chapter "AOCE Utilities" in this book.

kDETcmdSetPropertyChanged

This callback routine sets or clears the property-changed flag for a specified property.

```
struct DETSetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyChanged`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`propertyChanged`	`Boolean`	Property-changed flag

DESCRIPTION

If you set the `propertyChanged` field to `true`, the Catalogs Extension saves the property the next time the user closes the information page. Note that setting the value of a property does not automatically set its changed flag.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyChanged` callback routine (page 5-221) to determine the current value of a property's changed flag.

kDETcmdSetPropertyEditable

This callback routine sets the property-editable flag for a specific property.

```
struct DETSetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyEditable`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`propertyEditable`	`Boolean`	Property-editable flag

DESCRIPTION

The Catalogs Extension normally sets the value of the property-editable flag for a property based on the user's authentication identity and the access control settings of the dNode (catalog folder), record, and attribute. The property-editable flag determines whether an edit-text view is editable or a control in an information page is enabled. You can set the `propertyEditable` field to `false` to disable an edit-text view or a control, overriding the default setting, or to `true` to reenable a view or control once you have disabled it.

The setting of the `propertyEditable` flag persists only as long as the aspect remains in memory.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyEditable` callback routine (page 5-222) to determine the current setting of the property-editable flag.

kDETcmdDirtyProperty

This callback routine causes the CE to redraw a view and calls the code resource for the target with the `kDETcmdPropertyDirtied` routine selector.

```
struct DETDirtyPropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdDirtyProperty`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number

DESCRIPTION

When you make a change that affects the view associated with a property (by adding an item to a pop-up menu, for example), you can call the `kDETcmdDirtyProperty` callback routine to cause the Catalogs Extension to redraw the views associated with the property. This routine also calls the code resource for the target with the `kDETcmdPropertyDirtied` routine selector, giving you the opportunity to redraw other views affected by the views that were just redrawn.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

Calling the `kDETcmdDirtyProperty` routine does not cause the CE to save a property; when you change the value of a property, call the `kDETcmdSetPropertyChanged` routine (page 5-231) to cause the CE to save the new value.

kDETcmdSaveProperty

This callback routine saves the value of the specified property.

```
struct DETSavePropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSaveProperty`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number

DESCRIPTION

Normally, the Catalogs Extension saves all changed property values (that is, all property values for which the changed flag is set) when the user closes the information page. You can use the `kDETcmdSaveProperty` callback routine to force the CE to save a specific property immediately. The CE applies all the appropriate lookup-table patterns and writes the property values to the attributes specified by the lookup table.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Could not find or change property

SEE ALSO

Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

Working With Sublists

The routines in this section return information about sublists and force the Catalogs Extension to update a sublist. Note that the CE looks up information in catalogs asynchronously. Thus, it might not have finished setting up a sublist because it has not had time to complete its search. You can use the value of the property `kDETPastFirstLookup` to determine whether the CE has completed its catalog search. This property equals 0 until the search is complete, after which it equals 1.

kDETcmdSublistCount

This callback routine returns the number of items in the targeted aspect's sublist.

```
struct DETSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSublistCount`
→	`target`	`DETTargetSpecification`	Target specifier
←	`count`	`long`	The number of items in the targeted aspect's sublist

DESCRIPTION

You can use this routine to determine the total number of items in your aspect's sublist when you are using a targeted callback routine to iterate through every item in the sublist.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The target-specifier structure requires you to specify the index number of a sublist item. The target specifier is described in "Target Specifier" on page 5-142.

Use the `kDETcmdSelectedSublistCount` callback routine (described next) to determine the number of selected items in the sublist.

kDETcmdSelectedSublistCount

This callback routine returns the number of items that the user has selected in the targeted aspect's sublist.

```
struct DETSelectedSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSelectedSublistCount`
→	`target`	`DETTargetSpecification`	Target specifier
←	`count`	`long`	The number of selected items in the targeted aspect's sublist

DESCRIPTION

You can use this routine to determine the number of selected items in your aspect's sublist when you are using a targeted callback routine to iterate through all the selected items in the sublist.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The target-specifier structure requires you to specify the index number of a sublist item. The target specifier is described in "Target Specifier" on page 5-142.

Use the `kDETcmdSublistCount` callback routine (page 5-235) to determine the total number of items in the sublist.

kDETcmdRequestSync

This callback routine causes the CE to synchronize a sublist and properties with the catalog system.

```
struct DETRequestSyncBlock {
            DETCallBackBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdRequestSync`
→	`target`	`DETTargetSpecification`	Target specifier

DESCRIPTION

This routine forces the Catalogs Extension to check immediately whether the sublist or any properties in the targeted aspect need updating to match what's present in the catalog system. Normally, the CE performs this operation periodically. You can use this callback routine if you need the synchronization done immediately; for example, if you use a Catalog Manager function to add something to the sublist and want it displayed without delay.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

When you call the kDETcmdRequestSync callback routine, the CE calls your kDETcmdShouldSync routine (page 5-185) to determine whether any of your properties need updating.

Working With Pop-Up Menus

The commands in this section add and remove dynamic pop-up menu items and return the text of a pop-up menu item.

kDETcmdAddMenu

This callback routine adds an item to a dynamic pop-up menu.

```
struct DETAddMenuBlock {
            DETCallBackBlockPropertyHeader
            RString* name;
            long parameter;
            long addAfter;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdAddMenu
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	name	RString*	Pointer to name of new menu item
→	parameter	long	Parameter to return to code resource when this item is selected
→	addAfter	long	Parameter of menu item to add this item after, or –1 to add item at end of menu

DESCRIPTION

Provide a pointer to the text for the new menu item in the `name` field. The Catalogs Extension sends the value in the `parameter` field to your code resource as a parameter to the `kDETcmdPropertyCommand` routine when the user chooses this menu item. Use the `addAfter` parameter to indicate where in the menu to add the item: immediately after the menu item whose parameter you specify, or at the end of the menu if you specify –1.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

Pop-up menus are limited to 31 items. If you try to add more than 31 items, the `kDETcmdAddMenu` callback routine returns the `kDETCouldNotAddMenuItem` result code.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open
`kDETNoSuchView`	–15013	No view found with specified property number
`kDETCouldNotAddMenuItem`	–15014	Could not add item to menu
`kDETCouldNotFindMenuItem`	–15016	Could not find menu item

SEE ALSO

Use the `kDETcmdDirtyProperty` callback routine (page 5-233) to cause the CE to redraw the menu when you add a new menu item.

Pop-up menus are described in "View Lists" beginning on page 5-123.

kDETcmdRemoveMenu

This callback routine removes an item from a dynamic pop-up menu.

```
struct DETRemoveMenuBlock {
            DETCallBackBlockPropertyHeader
            long itemToRemove;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdRemoveMenu`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`itemToRemove`	`long`	Parameter of menu item to remove

DESCRIPTION

This routine removes the item that has the parameter value specified in the `itemToRemove` field.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open

kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotRemoveMenuItem	–15015	Could not remove item from dynamic menu
kDETCouldNotFindMenuItem	–15016	Could not find menu item

SEE ALSO

Use the kDETcmdDirtyProperty callback routine (page 5-233) to cause the CE to redraw the menu when you remove a menu item.

Pop-up menus are described in "View Lists" beginning on page 5-123.

kDETcmdMenuItemRString

This callback routine returns the text of an item in a dynamic pop-up menu.

```
struct DETMenuItemRStringBlock {
            DETCallBackBlockPropertyHeader
            long itemParameter;
            RStringHandle rString;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdMenuItemRString
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	itemParameter	long	Parameter of menu item for which you want the text string
←	rString	RStringHandle	Handle to string containing text of menu item

DESCRIPTION

Use the itemParameter field to specify the parameter of the menu item whose text you want. The Catalogs Extension allocates the handle for the rString field. It is your responsibility to deallocate the handle when it is no longer needed.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open
`kDETNoSuchView`	–15013	No view found with specified property number
`kDETCouldNotFindMenuItem`	–15016	Could not find menu item

SEE ALSO

Pop-up menus are described in "View Lists" beginning on page 5-123.

Custom Views

The routines in this section return information about custom views. The first routine, `kDETcmdGetCustomViewUserReference`, returns the reference value associated with a custom view. The `kDETcmdGetCustomViewBounds` routine returns the bounds for a custom view.

kDETcmdGetCustomViewUserReference

This callback routine returns the reference value that you set in the view list for a custom view.

```
struct DETGetCustomViewUserReferenceBlock {
            DETCallBackBlockPropertyHeader
            short userReference;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCustomViewUserReference
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	userReference	short	User reference value

DESCRIPTION

The view list specification for a custom view includes an integer that you can set to any value you wish. The `kDETcmdGetCustomViewUserReference` callback routine returns this value for a specific custom view.

SPECIAL CONSIDERATIONS

You cannot call this routine for a custom view that is not visible: the information page must be open and, if the custom view is in a conditional view, that view must be currently drawn.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open
kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotFindCustomView	–15017	Could not find custom view

SEE ALSO

The view list specifier for a custom view is described in "View Lists" beginning on page 5-123.

For more information about how to implement custom views, see "Custom Views and Custom Menus" beginning on page 5-192.

kDETcmdGetCustomViewBounds

This callback routine returns the bounds of a custom view.

```
struct DETGetCustomViewBoundsBlock {
            DETCallBackBlockPropertyHeader
            Rect bounds;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetCustomViewBounds`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`bounds`	`rect`	Bounds of the view in local window coordinates

DESCRIPTION

You can use this routine to determine the bounds of a specific custom view so that you don't have to store the bounds for every custom view you define.

SPECIAL CONSIDERATIONS

You cannot call this routine for a custom view that is not visible: the information page must be open and, if the custom view is in a conditional view, that view must be currently drawn.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open
`kDETNoSuchView`	–15013	No view found with specified property number
`kDETCouldNotFindCustomView`	–15017	Could not find custom view

SEE ALSO

The view list specifier for a custom view is described in "View Lists" beginning on page 5-123.

For more information about how to implement custom views, see "Custom Views and Custom Menus" beginning on page 5-192.

Sending a Property Command

The `kDETcmdDoPropertyCommand` callback routine sends a property command to a code resource.

kDETcmdDoPropertyCommand

This callback routine sends a property command to the targeted code resource.

```
struct DETDoPropertyCommandBlock {
            DETCallBackBlockPropertyHeader
            long parameter;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdDoPropertyCommand`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`parameter`	`long`	Parameter of property command

DESCRIPTION

When you call this routine, the Catalogs Extension calls your code resource's property command (`kDETcmdPropertyCommand`) routine. The CE passes the property number and parameter value you specify to your property command. The effect is the same as when the CE initiates a property command.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

The `kDETcmdPropertyCommand` routine is described on page 5-159.

Summary of AOCE Templates

C Summary

Constants and Data Types

```
/* Current versions of all the different template types */

#define kDETAspectVersion     -976
#define kDETInfoPageVersion   -976
#define kDETKillerVersion     -976
#define kDETForwarderVersion  -976
#define kDETFileTypeVersion   -976

/* Suggested separation for template IDs within a file */
#define kDETIDSep             250

/* Predefined base IDs */
#define kDETFirstID           (1000)
#define kDETSecondID          (1000 + kDETIDSep)
#define kDETThirdID           (1000 + 2 * kDETIDSep)
#define kDETFourthID          (1000 + 3 * kDETIDSep)
#define kDETFifthID           (1000 + 4 * kDETIDSep)

/* Template resource ID offsets */
#define kDETTemplateName               0
#define kDETRecordType                 1
#define kDETKillerName                 1
#define kDETAttributeType              2
#define kDETAttributeValueTag          3
#define kDETAspectCode                 4
#define kDETInfoPageName               4
#define kDETForwarderTemplateNames     4
#define kDETAspectMainBitmap           5
#define kDETInfoPageMainViewAspect     5
#define kDETAspectName                 6
#define kDETInfoPageMenuName           6
#define kDETAspectCategory             7
#define kDETInfoPageMenuEntries        7
```

```
#define kDETAspectExternalCategory     8
#define kDETAspectKind                 9
#define kDETAspectGender               10
#define kDETAspectWhatIs               11
#define kDETAspectAliasKind            12
#define kDETAspectAliasGender          13
#define kDETAspectAliasWhatIs          14
#define kDETAspectBalloons             15
#define kDETAspectNewMenuName          16
#define kDETAspectNewEntryName         17
#define kDETAspectNewValue             18
#define kDETAspectSublistOpenOnNew     19
#define kDETAspectLookup               20
#define kDETAspectDragInString         21
#define kDETAspectDragInVerb           22
#define kDETAspectDragInSummary        23
#define kDETAspectRecordDragIn         24
#define kDETAspectRecordCatDragIn      25
#define kDETAspectAttrDragIn           26
#define kDETAspectAttrDragOut          27
#define kDETAspectViewMenu             28
#define kDETAspectReverseSort          29
#define kDETAspectInfoPageCustomWindow 30

/* Properties */
#define kDETNoProperty                 -1
#define kDETFirstLocalProperty         0
#define kDETLastLocalProperty          (kDETFirstLocalProperty + 249)
#define kDETFirstDevProperty           40
#define kDETFirstConstantProperty      250
#define kDETLastConstantProperty       (kDETFirstConstantProperty + 249)
#define kDETConstantProperty           kDETFirstConstantProperty
#define kDETZeroProperty               (kDETConstantProperty + 0)
#define kDETOneProperty                (kDETConstantProperty + 1)
#define kDETFalseProperty              (kDETConstantProperty + 0)
#define kDETTrueProperty               (kDETConstantProperty + 1)

/* Name and kind properties */
#define kDETPrName                     3050
#define kDETPrKind                     3051

#define kDETPastFirstLookup         26550
#define kDETInfoPageNumber          27050
#define kDETAspectTemplateNumber    26551
```

```
#define kDETInfoPageTemplateNumber  26552
#define kDETOpenSelectedItems       26553 /* open selected sublist items */
#define kDETAddNewItem              26554 /* add new sublist item */
#define kDETRemoveSelectedItems     26555 /* remove selected sublist items */

/* Access masks */
#define kDETDNodeAccessMask         25825 /* the DNode access mask */
#define kDETRecordAccessMask        25826 /* the record access mask */
#define kDETAttributeAccessMask     25827 /* the attribute access mask */
#define kDETPrimaryMaskByBit        25828 /* a set of 16 properties
                                             to access all bits of the
                                             primary mask */

#define kDETPrimarySeeMask          kDETPrimaryMaskByBit
#define kDETPrimaryAddMask          (kDETPrimaryMaskByBit + 1)
#define kDETPrimaryDeleteMask       (kDETPrimaryMaskByBit + 2)
#define kDETPrimaryChangeMask       (kDETPrimaryMaskByBit + 3)
#define kDETPrimaryRenameMask       (kDETPrimaryMaskByBit + 4)
#define kDETPrimaryChangePrivsMask  (kDETPrimaryMaskByBit + 5)
#define kDETPrimaryTopMaskBit       (kDETPrimaryMaskByBit + 15)

/* Property types */
#define kDETPrTypeNumber            -1    /* a number */
#define kDETPrTypeString            -2    /* a string */
#define kDETPrTypeBinary            -3    /* a binary block */

/* Rez-compatible attribute-tag types */
#define typeRString                 'rstr'
#define typePackedDSSpec            'dspc'
#define typeBinary                  'bnry'

/* Constants used in view lists */
#define kDETNoFlags           0
#define kDETEnabled           (1 << 0) /* main view field enabled */

#define kDETHilightIfSelected (1 << 0) /* hilight when entry is selected */

#define kDETNumericOnly       (1 << 3) /* allow digits only */
#define kDETMultiLine         (1 << 4) /* allow multiple lines in view */
#define kDETDynamicSize       (1 << 9) /* don't draw box around text
                                          until user clicks in it,
                                          then auto-size it */
#define kDETAllowNoColons     (1 << 10)/* don't allow colons */
```

```
#define kDETPopupDynamicSize  (1 << 8) /* automatically resize pop-up */

#define kDETScaleToView       (1 << 8) /* scale picture to view bounds */

#define kDETLargeIcon              0
#define kDETSmallIcon              1
#define kDETMiniIcon               2

#define kDETLeft                   0
#define kDETCenter                 1
#define kDETRight                  -1
#define kDETForceLeft              -2

#define kDETUnused                 0
#define kDETBoxTakesContentClicks  (1 << 0)
#define kDETBoxIsRounded           (1 << 1)
#define kDETBoxIsGrayed            (1 << 2)
#define kDETBoxIsInvisible         (1 << 3)

#define kDETApplicationFont         1
#define kDETApplicationFontSize     9
#define kDETAppFontLineHeight      12

#define kDETSystemFont              0
#define kDETSystemFontSize         12
#define kDETSystemFontLineHeight   16

#define kDETDefaultFont             1
#define kDETDefaultFontSize         9
#define kDETDefaultFontLineHeight  12

#define kDETNormal                 0
#define kDETBold                   1
#define kDETItalic                 2
#define kDETUnderline              4
#define kDETOutline                8
#define kDETShadow                 0x10
#define kDETCondense               0x20
#define kDETExtend                 0x40

#define kDETIconStyle              -3 /* normal text style for
                                         regular sublist entries,
```

```
                                        italic text style for aliases */

#define kDETChangeViewCommand       'view'   /* change the view */

/* Default information-page layouts */

/* Default record information-page size */
#define kDETRecordInfoWindHeight    228
#define kDETRecordInfoWindWidth     400

/* Default attribute information-page size */
#define kDETAttributeInfoWindHeight 250
#define kDETAttributeInfoWindWidth  230

/* Page identifying icon */
#define kDETSubpageIconTop          8
#define kDETSubpageIconLeft         8
#define kDETSubpageIconBottom       (kDETSubpageIconTop + 32)
#define kDETSubpageIconRight        (kDETSubpageIconLeft + 32)
#define kDETSubpageIconRect         {kDETSubpageIconTop,\
                                     kDETSubpageIconLeft,\
                                     kDETSubpageIconBottom,\
                                     kDETSubpageIconRight}

/* The following rectangle can be used in a 'deti' with no sublist: */
#define kDETNoSublistRect           {0, 0, 0, 0}

/* Reserved category names */
#define kDETCategoryAllItems     "aoce All Items"     /* everything */
#define kDETCategoryAddressItems "aoce Address Items" /* all addresses */
#define kDETCategoryMisc         "aoce Miscellaneous" /* things that
                                         don't have their own category */

/* Target selectors */
enum DETTargetSelector {
   kDETSelf = 0,              /* the "current" item */
   kDETSelfOtherAspect,       /* another aspect of the current item */
   kDETParent,                /* the parent of the current item */
   kDETSublistItem,           /* the ith item in the sublist */
   kDETSelectedSublistItem,   /* the ith selected item in the sublist */
   kDETDSSpec,                /* DSSpec */
   kDETAspectTemplate,        /* specific aspect template */
   kDETInfoPageTemplate,      /* specific info-page template */
   kDETHighSelector = 0xF000  /* force type to be short */
```

```
};

typedef enum DETTargetSelector DETTargetSelector;

/* Return value for code resources */
#define kDETDidNotHandle 1

/* Valid commandIDs for DETDropQueryBlock and DETDropMeQueryBlock (in
    addition to property numbers) */
#define kDETDoNothing   'xxx0'
#define kDETMove        'move'
#define kDETDrag        'drag'
#define kDETAlias       'alis'

/* Item types */
enum DETItemType {
   kDETHFSType = 0,             /* HFS item type */
   kDETDSType,                  /* catalog service item type */
   kDETMailType,                /* mail (letter) item type */
   kDETMoverType,               /* sounds, fonts, etc., from inside
                                    a suitcase or system file */
   kDETLastItemType = 0xF0000000 /* force itemType to be a long */
};

typedef enum DETItemType DETItemType;

struct DETFSInfo {
   OSType fileType;        /* file type */
   OSType fileCreator;     /* file creator */
   unsigned short fdFlags; /* Finder flags */
   FSSpec fsSpec;          /* FSSpec */
};

typedef struct DETFSInfo DETFSInfo;

struct {
   PackedDSSpecPtr* dsSpec;   /* DSSpec for item */
   short refNum;              /* refnum for returned address */
   AuthIdentity identity;     /* identity for returned address */
} ds;

/* Application-defined routines */
enum DETCallFunctions {

   kDETcmdSimpleCall = 0,
```

```
    kDETcmdInit,
    kDETcmdExit,
    kDETcmdAttributeCreation,
    kDETcmdDynamicForwarders,

    kDETcmdTargetedCall = 1000,
    kDETcmdInstanceInit,
    kDETcmdInstanceExit,
    kDETcmdIdle,
    kDETcmdViewListChanged,
    kDETcmdValidateSave,
    kDETcmdDropQuery,
    kDETcmdDropMeQuery,
    kDETcmdAttributeNew,
    kDETcmdAttributeChange,
    kDETcmdAttributeDelete,
    kDETcmdItemNew,
    kDETcmdOpenSelf,
    kDETcmdDynamicResource,
    kDETcmdShouldSync,
    kDETcmdDoSync,

    kDETcmdPropertyCall = 2000,
    kDETcmdPropertyCommand,
    kDETcmdMaximumTextLength,
    kDETcmdPropertyDirtied,
    kDETcmdPatternIn,
    kDETcmdPatternOut,
    kDETcmdConvertToNumber,
    kDETcmdConvertToRString,
    kDETcmdConvertFromNumber,
    kDETcmdConvertFromRString,
    kDETcmdCustomViewDraw,
    kDETcmdCustomViewMouseDown,
    kDETcmdKeyPress,
    kDETcmdPaste,
    kDETcmdCustomMenuSelected,
    kDETcmdCustomMenuEnabled,

    kDETcmdHighCall = 0xF0000000/* force the type to be long */
};

typedef enum DETCallFunctions DETCallFunctions;
```

```
/* Callback functions */
enum DETCallBackFunctions {

   kDETcmdSimpleCallback = 0,
   kDETcmdBeep,
   kDETcmdBusy,
   kDETcmdChangeCallFors,
   kDETcmdGetCommandSelectionCount,
   kDETcmdGetCommandItemN,
   kDETcmdOpenDSSpec,
   kDETcmdAboutToTalk,
   kDETcmdUnloadTemplates,
   kDETcmdTemplateCounts,

   kDETcmdTargetedCallback = 1000,
   kDETcmdGetDSSpec,
   kDETcmdSublistCount,
   kDETcmdSelectedSublistCount,
   kDETcmdRequestSync,
   kDETcmdBreakAttribute,
   kDETcmdGetTemplateFSSpec,
   kDETcmdGetOpenEdit,
   kDETcmdCloseEdit,

   kDETcmdPropertyCallback = 2000,
   kDETcmdGetPropertyType,
   kDETcmdGetPropertyNumber,
   kDETcmdGetPropertyRString,
   kDETcmdGetPropertyBinarySize,
   kDETcmdGetPropertyBinary,
   kDETcmdGetPropertyChanged,
   kDETcmdGetPropertyEditable,
   kDETcmdSetPropertyType,
   kDETcmdSetPropertyNumber,
   kDETcmdSetPropertyRString,
   kDETcmdSetPropertyBinary,
   kDETcmdSetPropertyChanged,
   kDETcmdSetPropertyEditable,
   kDETcmdDirtyProperty,
   kDETcmdDoPropertyCommand,
   kDETcmdAddMenu,
   kDETcmdRemoveMenu,
   kDETcmdMenuItemRString,
```

```
   kDETcmdSaveProperty,
   kDETcmdGetCustomViewUserReference,
   kDETcmdGetCustomViewBounds,
   kDETcmdGetResource,

   kDETcmdHighCallback = 0xF0000000          /* force type to be long */
};

typedef enum DETCallBackFunctions DETCallBackFunctions;
```

Target Specifier

```
struct DETTargetSpecification
{
      DETTargetSelector selector;   /* target selector */
      RStringPtr aspectName;        /* aspect name */
      long itemNumber;              /* sublist index number */
      PackedDSSpecPtr dsSpec;       /* DSSpec */
};

typedef struct DETTargetSpecification DETTargetSpecification;
```

Forwarder List

```
struct DETForwarderListItem {
   struct DETForwarderListItem** next;/* handle to next item, or nil */
   AttributeTag attributeValueTag;    /* attribute value tag (0 for none) */
   PackedPathName rstrs;              /* forwarder list */
};
```

Call Block Headers

```
#define DETCallBlockHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */

#define DETCallBlockTargetedHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
```

```
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */

#define DETCallBlockPropertyHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */\
   short property;               /* the property number the call refers to */

struct DETProtoCallBlock {
   DETCallBlockPropertyHeader
};

typedef struct DETProtoCallBlock DETProtoCallBlock;
```

Call Block Union Structure

```
union DETCallBlock {
   DETProtoCallBlock               protoCall;
   DETInitBlock                    init;
   DETExitBlock                    exit;
   DETInstanceInitBlock            instanceInit;
   DETInstanceExitBlock            instanceExit;
   DETInstanceIdleBlock            instanceIdle;
   DETPropertyCommandBlock         propertyCommand;
   DETMaximumTextLengthBlock       maximumTextLength;
   DETViewListChangedBlock         viewListChanged;
   DETPropertyDirtiedBlock         propertyDirtied;
   DETValidateSaveBlock            validateSave;
   DETDropQueryBlock               dropQuery;
   DETDropMeQueryBlock             dropMeQuery;
   DETAttributeCreationBlock       attributeCreationBlock;
   DETAttributeNewBlock            attributeNew;
   DETAttributeChangeBlock         attributeChange;
   DETAttributeDeleteBlock         attributeDelete;
   DETItemNewBlock                 itemNew;
   DETPatternInBlock               patternIn;
   DETPatternOutBlock              patternOut;
   DETShouldSyncBlock              shouldSync;
```

```
   DETDoSyncBlock                  doSync;
   DETOpenSelfBlock                openSelf;
   DETConvertToNumberBlock         convertToNumber;
   DETConvertToRStringBlock        convertToRString;
   DETConvertFromNumberBlock       convertFromNumber;
   DETConvertFromRStringBlock      convertFromRString;
   DETCustomViewDrawBlock          customViewDraw;
   DETCustomViewMouseDownBlock     customViewMouseDown;
   DETKeyPressBlock                keyPress;
   DETPasteBlock                   paste;
   DETCustomMenuSelectedBlock      customMenuSelected;
   DETCustomMenuEnabledBlock       customMenuEnabled;
   DETDynamicForwardersBlock       dynamicForwarders;
   DETDynamicResourceBlock         dynamicResource;
};

typedef union DETCallBlock DETCallBlock;
typedef DETCallBlock* DETCallBlockPtr;
```

Callback Block Headers

```
#define DETCallBackBlockHeader \
   DETCallBackFunctions reqFunction;   /* requested function */

#define DETCallBackBlockTargetedHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */

#define DETCallBackBlockPropertyHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */\
   short property;                     /* the property to apply the
                                          request to */

struct DETProtoCallBackBlock {
   DETCallBackBlockPropertyHeader
};

typedef struct DETProtoCallBackBlock DETProtoCallBackBlock;
```

Callback Block Union Structure

```
union DETCallBackBlock {
   DETProtoCallBackBlock                 protoCallBack;
   DETBeepBlock                          beep;
   DETBusyBlock                          busy;
   DETChangeCallForsBlock                changeCallFors;
   DETGetCommandSelectionCountBlock      getCommandSelectionCount;
   DETGetCommandItemNBlock               getCommandItemN;
   DETGetDSSpecBlock                     getDSSpec;
   DETGetTemplateFSSpecBlock             getTemplateFSSpec;
   DETGetOpenEditBlock                   getOpenEdit;
   DETCloseEditBlock                     closeEdit;
   DETGetPropertyTypeBlock               getPropertyType;
   DETGetPropertyNumberBlock             getPropertyNumber;
   DETGetPropertyRStringBlock            getPropertyRString;
   DETGetPropertyBinarySizeBlock         getPropertyBinarySize;
   DETGetPropertyBinaryBlock             getPropertyBinary;
   DETGetPropertyChangedBlock            getPropertyChanged;
   DETGetPropertyEditableBlock           getPropertyEditable;
   DETSetPropertyTypeBlock               setPropertyType;
   DETSetPropertyNumberBlock             setPropertyNumber;
   DETSetPropertyRStringBlock            setPropertyRString;
   DETSetPropertyBinaryBlock             setPropertyBinary;
   DETSetPropertyChangedBlock            setPropertyChanged;
   DETSetPropertyEditableBlock           setPropertyEditable;
   DETDirtyPropertyBlock                 dirtyProperty;
   DETDoPropertyCommandBlock             doPropertyCommand;
   DETSublistCountBlock                  sublistCount;
   DETSelectedSublistCountBlock          selectedSublistCount;
   DETRequestSyncBlock                   requestSync;
   DETAddMenuBlock                       addMenu;
   DETRemoveMenuBlock                    removeMenu;
   DETMenuItemRStringBlock               menuItemRString;
   DETOpenDSSpecBlock                    openDSSpec;
   DETAboutToTalkBlock                   aboutToTalk;
   DETBreakAttributeBlock                breakAttribute;
   DETSavePropertyBlock                  saveProperty;
   DETGetCustomViewUserReferenceBlock    getCustomViewUserReference;
   DETGetCustomViewBoundsBlock           getCustomViewBounds;
   DETGetResourceBlock                   getResource;
   DETTemplateCounts                     templateCounts;
   DETUnloadTemplatesBlock               unloadTemplates;
};
```

```
typedef union DETCallBackBlock DETCallBackBlock;
typedef DETCallBackBlock* DETCallBackBlockPtr;

typedef pascal OSErr (*DETCallBack) (union DETCallBlock* callBlockPtr,
                                     DETCallBackBlockPtr callBackBlockPtr);
```

Call-For Mask

```
/* Call-for list: */

#define kDETCallForOther         1     /* call for events not listed below */
#define kDETCallForIdle          2     /* kDETcmdIdle */
#define kDETCallForCommands      4     /* kDETcmdPropertyCommand,
                                          kDETcmdSelfOpen */
#define kDETCallForViewChanges   8     /* kDETcmdViewListChanged
                                          kDETcmdPropertyDirtied,
                                          kDETcmdMaximumTextLength */
#define kDETCallForDrops         0x10  /* kDETcmdDropQuery,
                                          kDETcmdDropMeQuery */
#define kDETCallForAttributes    0x20  /* kDETcmdAttributeCreation,
                                          kDETcmdAttributeNew,
                                          kDETcmdAttributeChange,
                                          kDETcmdAttributeDelete */
#define kDETCallForValidation    0x40  /* kDETcmdValidateSave */
#define kDETCallForKeyPresses    0x80  /* kDETcmdKeyPress and
                                          kDETcmdPaste */
#define kDETCallForSyncing       0x200 /* kDETcmdShouldSync, kDETcmdDoSync */
#define kDETCallForResources     0x100 /* kDETcmdDynamicResource */
#define kDETCallForEscalation    0x8000/* all calls escalated to the
                                          next level */

#define kDETCallForNothing       0     /* do not call */
#define kDETCallForEverything    0xFFFFFFFF  /* all of the above */

typedef pascal OSErr (*DETCall) (DETCallBlockPtr callBlockPtr);
```

Functions You Can Provide as Part of Your Code Resource

Initializing and Removing Templates

```
struct DETInitBlock {
            DETCallBlockHeader
            long newCallFors;
};

struct DETExitBlock{
            DETCallBlockHeader
};

struct DETInstanceInitBlock {
            DETCallBlockTargetedHeader
};

struct DETItemNewBlock{
            DETCallBlockTargetedHeader
};

struct DETInstanceExitBlock {
            DETCallBlockTargetedHeader
};
```

Dynamic Creation of Resources

```
struct DETDynamicForwardersBlock {
            DETCallBlockHeader
            DETForwarderListHandle forwarders;
};

struct DETDynamicResourceBlock {
            DETCallBlockTargetedHeader
            ResType resourceType;
            short propertyNumber;
            short resourceID;
            Handle theResource;
};
```

Processing Idle-Time Tasks

```
struct DETcmdInstanceIdleBlock {
            DETCallBlockTargetedHeader
};
```

Property and Information Page Routines

```
struct DETOpenSelfBlock {
            DETCallBlockTargetedHeader
            short modifiers;
};

struct DETPropertyCommandBlock {
            DETCallBlockPropertyHeader
            long parameter;
};

struct DETKeyPressBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};

struct DETPasteBlock {
            DETCallBlockPropertyHeader
            short modifiers;
};

struct DETMaximumTextLengthBlock {
            DETCallBlockPropertyHeader
            long MaxSize;
};

struct DETViewListChangedBlock {
            DETCallBlockTargetedHeader
};

struct DETPropertyDirtiedBlock {
            DETCallBlockPropertyHeader
};

struct DETValidateSaveBlock {
            DETCallBlockTargetedHeader
            RStringHandle errorString;
};
```

Supporting Drops

```
struct DETDropMeQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
```

```
            long commandID;
            AttributeType destinationType
            Boolean copyToHFS;
};

struct DETDropQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType
            Boolean copyToHFS;
};
```

Attribute-Related Commands

```
struct DETAttributeCreationBlock {
            DETCallBlockHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};

struct DETAttributeNewBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};

struct DETAttributeChangeBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
```

```
            AttributeCreationID attrCID;
            Handle value;
};

struct DETAttributeDeleteBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr dsSpec;
            short refNum;
            AuthIdentity identity;
};
```

Processing Custom Lookup-Table Pattern Elements

```
struct DETPatternInBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            long dataOffset;
            short bitOffset;
};

struct DETPatternOutBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            Handle data;
            long dataOffset;
            short bitOffset;
};
```

Synchronizing Property Values

```
struct DETShouldSyncBlock {
            DETCallBlockTargetedHeader
            Boolean shouldSync;
};

struct DETDoSyncBlock {
            DETCallBlockTargetedHeader;
};
```

Custom Property-Type Conversions

```
struct DETConvertToNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};

struct DETConvertToRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};

struct DETConvertFromNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};

struct DETConvertFromRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Custom Views and Custom Menus

```
struct DETGetCustomViewDrawBlock {
            DETCallBlockPropertyHeader
};

struct DETCustomViewMouseDownBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};

struct DETCustomMenuEnabledBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter
            Boolean enable;
};

struct DETCustomMenuSelectedBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
);
```

CE-Provided Functions That Your Code Resource Can Call

Calling CE-Provided Functions

```
CallBackDET(callBlockPtr, callBackBlockPtr);
```

Testing Your Code Resource

```
struct DETBeepBlock {
            DETCallBackBlockHeader
};
```

Changing the Call-For Mask

```
struct DETChangeCallForsBlock {
            DETCallBackBlockTargetedHeader
            long newCallFors;
};
```

Process Control

```
struct DETAboutToTalkBlock {
            DETCallBackBlockHeader
};

struct DETBusyBlock {
            DETCallBackBlockHeader;
};
```

Handling Drags and Drops

```
struct DETGetCommandSelectionCountBlock {
            DETCallBackBlockHeader;
            long count;
};

struct DETGetCommandItemNBlock {
      DETCallBackBlockHeader;
      long itemNumber;
      DETItemType itemType;
      union {
         DETFSInfo** fsInfo;
         struct {
                  PackedDSSpecPtr* dsSpec;
```

```
                short refNum;
                AuthIdentity identity;
          } ds;
          PackedDSSpecPtr* dsSpec;
          LetterSpec** ltrSpec;
       } item;
};
```

Working With Templates

```
struct DETTemplateCounts {
            DETCallBackBlockHeader
            long aspectTemplateCount;
            long infoPageTemplateCount;
};

struct DETGetTemplateFSSpecBlock {
            DETCallBackBlockTargetedHeader
            FSSpec fsSpec;
            short baseID;
            long aspectTemplateNumber;
};

struct DETGetResourceBlock {
            DETCallBackBlockPropertyHeader
            ResType resourceType;
            Handle theResource;
};

struct DETUnloadTemplatesBlock {
   DETCallBackBlockHeader
};
```

Working With Catalog Objects

```
struct DETGetDSSpecBlock {
            DETCallBackBlockTargetedHeader
            PackedDSSpecPtr* dsSpec;
            short refNum;
            AuthIdentity identity;
            Boolean isAlias;
            Boolean isRecordRef;
};
```

```
struct DETOpenDSSpecBlock {
            DETCallBackBlockHeader
            PackedDSSpecPtr dsSpec;
};
```

Edit-Text Routines

```
struct DETGetOpenEditBlock {
            DETCallBackBlockTargetedHeader
            short viewProperty;
};

struct DETCloseEditBlock {
            DETCallBackBlockTargetedHeader
};
```

Getting Information About Properties

```
struct DETGetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short propertyType;
};

struct DETGetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long propertyValue;
};

struct DETGetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringHandle propertyValue;
};

struct DETGetPropertyBinarySizeBlock {
            DETCallBackBlockPropertyHeader
            long propertyBinarySize;
};

struct DETGetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Handle propertyValue;
};
```

```
struct DETGetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};

struct DETGetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Setting Value, Type, and Other Features of Properties

```
struct DETBreakAttributeBlock {
            DETCallBackBlockTargetedHeader
            AttributePtr breakAttribute;
            Boolean isChangeable;
};

struct DETSetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short newType;
};

struct DETSetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long newValue;
};

struct DETSetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringPtr newValue;
};

struct DETSetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Ptr newValue;
            long newValueSize;
};

struct DETSetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

```
struct DETSetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};

struct DETDirtyPropertyBlock {
            DETCallBackBlockPropertyHeader
};

struct DETSavePropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Working With Sublists

```
struct DETSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};

struct DETSelectedSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};

struct DETRequestSyncBlock {
            DETCallBackBlockTargetedHeader
};
```

Working With Pop-Up Menus

```
struct DETAddMenuBlock {
            DETCallBackBlockPropertyHeader
            RString* name;
            long parameter;
            long addAfter;
};

struct DETRemoveMenuBlock {
            DETCallBackBlockPropertyHeader
            long itemToRemove;
};
```

```
struct DETMenuItemRStringBlock {
            DETCallBackBlockPropertyHeader
            long itemParameter;
            RStringHandle rString;
};
```

Custom Views

```
struct DETGetCustomViewUserReferenceBlock {
            DETCallBackBlockPropertyHeader
            short userReference;
};

struct DETGetCustomViewBoundsBlock {
            DETCallBackBlockPropertyHeader
            Rect bounds;
};
```

Sending a Property Command

```
struct DETDoPropertyCommandBlock {
            DETCallBackBlockPropertyHeader
            long parameter;
};
```

Pascal Summary

Constants

```
{Current versions of all the different template types}

CONST
kDETAspectVersion             = -976;
kDETInfoPageVersion           = -976;
kDETKillerVersion             = -976;
kDETForwarderVersion          = -976;
kDETFileTypeVersion           = -976;

{Suggested separation for template IDs within a file}
kDETIDSep                     = 250
```

```
{Predefined base IDs}
kDETFirstID                  = (1000);
kDETSecondID                 = (1000 + kDETIDSep);
kDETThirdID                  = (1000 + 2 * kDETIDSep);
kDETFourthID                 = (1000 + 3 * kDETIDSep);
kDETFifthID                  = (1000 + 4 * kDETIDSep);

{Template resource ID offsets}
kDETTemplateName                     = 0;
kDETRecordType                       = 1;
kDETKillerName                       = 1;
kDETAttributeType                    = 2;
kDETAttributeValueTag                = 3;
kDETAspectCode                       = 4;
kDETInfoPageName                     = 4;
kDETForwarderTemplateNames           = 4;
kDETAspectMainBitmap                 = 5;
kDETInfoPageMainViewAspect           = 5;
kDETAspectName                       = 6;
kDETInfoPageMenuName                 = 6;
kDETAspectCategory                   = 7;
kDETInfoPageMenuEntries              = 7;
kDETAspectExternalCategory           = 8;
kDETAspectKind                       = 9;
kDETAspectGender                     = 10;
kDETAspectWhatIs                     = 11;
kDETAspectAliasKind                  = 12;
kDETAspectAliasGender                = 13;
kDETAspectAliasWhatIs                = 14;
kDETAspectBalloons                   = 15;
kDETAspectNewMenuName                = 16;
kDETAspectNewEntryName               = 17;
kDETAspectNewValue                   = 18;
kDETAspectSublistOpenOnNew           = 19;
kDETAspectLookup                     = 20;
kDETAspectDragInString               = 21;
kDETAspectDragInVerb                 = 22;
kDETAspectDragInSummary              = 23;
kDETAspectRecordDragIn               = 24;
kDETAspectRecordCatDragIn            = 25;
kDETAspectAttrDragIn                 = 26;
kDETAspectAttrDragOut                = 27;
```

```
kDETAspectViewMenu                  = 28;
kDETAspectReverseSort               = 29;
kDETAspectInfoPageCustomWindow      = 30;

{Properties};
kDETNoProperty                      -1;
kDETFirstLocalProperty              0;
kDETLastLocalProperty               (kDETFirstLocalProperty + 249);
kDETFirstDevProperty                40;
kDETFirstConstantProperty           250;
kDETLastConstantProperty            (kDETFirstConstantProperty + 249);
kDETConstantProperty                kDETFirstConstantProperty;
kDETZeroProperty                    (kDETConstantProperty + 0);
kDETOneProperty                     (kDETConstantProperty + 1);
kDETFalseProperty                   (kDETConstantProperty + 0);
kDETTrueProperty                    (kDETConstantProperty + 1);

{Name and kind properties}
kDETPrName                          3050;
kDETPrKind                          3051;

kDETPastFirstLookup             26550;
kDETInfoPageNumber              27050;
kDETAspectTemplateNumber        26551;
kDETInfoPageTemplateNumber      26552;
kDETOpenSelectedItems           26553;{open selected sublist items}
kDETAddNewItem                  26554;{add new sublist item}
kDETRemoveSelectedItems         26555;{remove selected sublist items}

{Access masks}
kDETDNodeAccessMask             25825;{the DNode access mask}
kDETRecordAccessMask            25826;{the record access mask}
kDETAttributeAccessMask         25827;{the attribute access mask}
kDETPrimaryMaskByBit            25828;{a set of 16 properties
                                          to access all bits of the
                                          primary mask}

kDETPrimarySeeMask              kDETPrimaryMaskByBit;
kDETPrimaryAddMask              (kDETPrimaryMaskByBit + 1);
kDETPrimaryDeleteMask           (kDETPrimaryMaskByBit + 2);
kDETPrimaryChangeMask           (kDETPrimaryMaskByBit + 3);
kDETPrimaryRenameMask           (kDETPrimaryMaskByBit + 4);
kDETPrimaryChangePrivsMask      (kDETPrimaryMaskByBit + 5);
kDETPrimaryTopMaskBit           (kDETPrimaryMaskByBit + 15);
```

```
{Property types}
kDETPrTypeNumber                    -1    {a number}
kDETPrTypeString                    -2    {a string}
kDETPrTypeBinary                    -3    {a binary block}

{Rez-compatible attribute-tag types}
typeRString                         'rstr';
typePackedDSSpec                    'dspc';
typeBinary                          'bnry';

{Constants used in view lists}
kDETNoFlags                   $0000;
kDETEnabled                   $0001;   {main view field enabled}

kDETHilightIfSelected         $0001;   {hilight when entry is selected}

kDETNumericOnly               $0008;   {allow digits only}
kDETMultiLine                 $0010;   {allow multiple lines in view}
kDETDynamicSize               $0200;   {don't draw box around text
                                          until user clicks in it,
                                          then auto-size it}
kDETAllowNoColons             $0400;   {don't allow colons}

kDETPopupDynamicSize          $0100;   {automatically resize pop-up}

kDETScaleToView               $0100;   {scale picture to view bounds}

kDETLargeIcon                       0;
kDETSmallIcon                       1;
kDETMiniIcon                        2;

kDETLeft                            0;
kDETCenter                          1;
kDETRight                           -1;
kDETForceLeft                       -2;

kDETUnused                          0;
kDETBoxTakesContentClicks           $0001;
kDETBoxIsRounded                    $0002;
kDETBoxIsGrayed                     $0004;
kDETBoxIsInvisible                  $0008;

kDETApplicationFont                  1;
kDETApplicationFontSize              9;
```

```
kDETAppFontLineHeight               12;

kDETSystemFont                       0;
kDETSystemFontSize                  12;
kDETSystemFontLineHeight            16;

kDETDefaultFont                      1;
kDETDefaultFontSize                  9;
kDETDefaultFontLineHeight           12;

kDETNormal                          0;
kDETBold                            1;
kDETItalic                          2;
kDETUnderline                       4;
kDETOutline                         8;
kDETShadow                          $10;
kDETCondense                        $20;
kDETExtend                          $40;

kDETIconStyle                       -3;   {normal text style for
                                            regular sublist entries,
                                            italic text style for aliases}

kDETChangeViewCommand               'view';  {change the view}

{Default information page layouts}

{Default record information page size}
kDETRecordInfoWindHeight            228;
kDETRecordInfoWindWidth             400;

{Default attribute information page size}
kDETAttributeInfoWindHeight         250;
kDETAttributeInfoWindWidth          230;

{Page identifying icon}
kDETSubpageIconTop                  8;
kDETSubpageIconLeft                 8;
kDETSubpageIconBottom               (kDETSubpageIconTop + 32);
kDETSubpageIconRight                (kDETSubpageIconLeft + 32);
*( #define kDETSubpageIconRect      (kDETSubpageIconTop,\
                                     kDETSubpageIconLeft,\
                                     kDETSubpageIconBottom,\
                                     kDETSubpageIconRight) *)
```

```
{The following rectangle can be used in a 'deti' with no sublist:}
(* #define kDETNoSublistRect        {0, 0, 0, 0} *)

{Reserved category names}
kDETCategoryAllItems             'aoce All Items';    {everything}
kDETCategoryAddressItems         'aoce Address Items';{all addresses}
kDETCategoryMisc                 'aoce Miscellaneous';{things that
                                          don't have their own category}

{Target selectors}
enum DETTargetSelector {
   kDETSelf = 0;                 {the "current" item}
   kDETSelfOtherAspect = 1;      {another aspect of the current item}
   kDETParent = 2;               {the parent of the current item}
   kDETSublistItem = 3;          {the ith item in the sublist}
   kDETSelectedSublistItem = 4;  {the ith selected item in the sublist}
   kDETDSSpec = 5;               {DSSpec}
   kDETAspectTemplate = 6;       {specific aspect template}
   kDETInfoPageTemplate = 7;     {specific info-page template}
   kDETHighSelector = $F000      {force type to be short}
};

{Return value for code resources}
CONST kDETDidNotHandle  = 1;

{Valid commandIDs for DETDropQueryBlock and DETDropMeQueryBlock (in
    addition to property numbers)}
CONST
   kDETDoNothing           = 'xxx0';
   kDETMove                = 'move';
   kDETDrag                = 'drag';
   kDETAlias               = 'alis';

{Application-defined routines}
CONST
   kDETcmdSimpleCall          = 0;
   kDETcmdInit                = 1;
   kDETcmdExit                = 2;
   kDETcmdAttributeCreation   = 3;
   kDETcmdDynamicForwarders   = 4;

   kDETcmdTargetedCall        = 1000;
   kDETcmdInstanceInit        = 1001;
```

```
   kDETcmdInstanceExit        = 1002;
   kDETcmdIdle                = 1003;
   kDETcmdViewListChanged     = 1004;
   kDETcmdValidateSave        = 1005;
   kDETcmdDropQuery           = 1006;
   kDETcmdDropMeQuery         = 1007;
   kDETcmdAttributeNew        = 1008;
   kDETcmdAttributeChange     = 1009;
   kDETcmdAttributeDelete     = 1010;
   kDETcmdItemNew             = 1011;
   kDETcmdOpenSelf            = 1012;
   kDETcmdDynamicResource     = 1013;
   kDETcmdShouldSync          = 1014;
   kDETcmdDoSync              = 1015;

   kDETcmdPropertyCall        = 2000
   kDETcmdPropertyCommand     = 2001;
   kDETcmdMaximumTextLength   = 2002;
   kDETcmdPropertyDirtied     = 2003 ;
   kDETcmdPatternIn           = 2004;
   kDETcmdPatternOut          = 2005;
   kDETcmdConvertToNumber     = 2006;
   kDETcmdConvertToRString    = 2007;
   kDETcmdConvertFromNumber   = 2008;
   kDETcmdConvertFromRString  = 2009;
   kDETcmdCustomViewDraw      = 2010;
   kDETcmdCustomViewMouseDown = 2011;
   kDETcmdKeyPress            = 2012;
   kDETcmdPaste               = 2013;
   kDETcmdCustomMenuSelected  = 2014;
   kDETcmdCustomMenuEnabled   = 2015;

   kDETcmdHighCall            = $F0000000    {force the type to be long}
};

{Callback functions}
CONST
   kDETcmdSimpleCallback           = 0;
   kDETcmdBeep                     = 1;
   kDETcmdBusy                     = 2;
   kDETcmdChangeCallFors           = 3;
   kDETcmdGetCommandSelectionCount = 4;
   kDETcmdGetCommandItemN          = 5;
   kDETcmdOpenDSSpec               = 6;
```

```
   kDETcmdAboutToTalk              = 7;
   kDETcmdUnloadTemplates          = 8;
   kDETcmdTemplateCounts           = 9;

   kDETcmdTargetedCallback         = 1000;
   kDETcmdGetDSSpec                = 1001;
   kDETcmdSublistCount             = 1002;
   kDETcmdSelectedSublistCount     = 1003;
   kDETcmdRequestSync              = 1004;
   kDETcmdBreakAttribute           = 1005;
   kDETcmdGetTemplateFSSpec        = 1006;
   kDETcmdGetOpenEdit              = 1007;
   kDETcmdCloseEdit                = 1008;

   kDETcmdPropertyCallback         = 2000;
   kDETcmdGetPropertyType          = 2001;
   kDETcmdGetPropertyNumber        = 2002;
   kDETcmdGetPropertyRString       = 2003;
   kDETcmdGetPropertyBinarySize    = 2004;
   kDETcmdGetPropertyBinary        = 2005;
   kDETcmdGetPropertyChanged       = 2006;
   kDETcmdGetPropertyEditable      = 2007;
   kDETcmdSetPropertyType          = 2008;
   kDETcmdSetPropertyNumber        = 2009;
   kDETcmdSetPropertyRString       = 2010;
   kDETcmdSetPropertyBinary        = 2011;
   kDETcmdSetPropertyChanged       = 2012;
   kDETcmdSetPropertyEditable      = 2013;
   kDETcmdDirtyProperty            = 2014;
   kDETcmdDoPropertyCommand        = 2015;
   kDETcmdAddMenu                  = 2016;
   kDETcmdRemoveMenu               = 2017;
   kDETcmdMenuItemRString          = 2018;
   kDETcmdSaveProperty             = 2019;
   kDETcmdGetCustomViewUserReference = 2020;
   kDETcmdGetCustomViewBounds      = 2021;
   kDETcmdGetResource              = 2022;

   kDETcmdHighCallback             = $F0000000;   {force type to be LongInt}

CONST
{Values of DETItemType}
kDETHFSType       = 0;          {HFS item type}
kDETDSType        = 1;          {Catalog Service item type}
```

```
kDETMailType      = 2;          {Mail (letter) item type}
kDETMoverType     = 3;          {sounds, fonts, etc., from inside a
                                   suitcase or system file}
kDETLastItemType  = $F0000000;  {force it to be a LongInt}
```

Call-For Mask

```
CONST
   kDETCallForOther        = 1;     {call for events not listed below}
   kDETCallForIdle         = 2;     {kDETcmdIdle}
   kDETCallForCommands     = 4;     {kDETcmdPropertyCommand, kDETcmdSelfOpen}
   kDETCallForViewChanges  = 8;     {kDETcmdViewListChanged,
                                       kDETcmdPropertyDirtied,
                                       kDETcmdMaximumTextLength}
   kDETCallForDrops        = $10;   {kDETcmdDropQuery, kDETcmdDropMeQuery}
   kDETCallForAttributes   = $20;   {kDETcmdAttributeCreation,
                                       kDETcmdAttributeNew,
                                       kDETcmdAttributeChange,
                                       kDETcmdAttributeDelete}
   kDETCallForValidation   = $40;   {kDETcmdValidateSave}
   kDETCallForKeyPresses   = $80;   {kDETcmdKeyPress, kDETcmdPaste}
   kDETCallForResources    = $100;  {kDETcmdDynamicResource}
   kDETCallForSyncing      = $200;  {kDETcmdShouldSync, kDETcmdDoSync}
   kDETCallForEscalation   = $8000; {all calls escalated to the next level}

   kDETCallForNothing      = 0;     {none of the above}
   kDETCallForEverything   = $FFFFFFFF;   {all of the above}
```

Data Types

```
TYPE
   DETTargetSelector = Integer;

   DETCallBackFunctions = LongInt;

   DETCallFunctions = LongInt;

   DETCall = ProcPtr;

   DETItemType = LongInt;

{FSSpec plus additional info}
   DETFSInfo =
   RECORD
```

```
      fileType: OSType;    {File type}
      fileCreator: OSType; {File creator}
      fdFlags: Integer;    {Finder flags}
      fsSpec: FSSpec;      {FSSpec}
   END;

DETFSInfoPtr = ^DETFSInfo;

LetterSpecPtr = ^LetterSpec;

LetterSpecHandle = ^LetterSpecPtr;
```

Target Specifier

```
TYPE
   DETTargetSelector = Integer

   DETTargetSpecification =
   RECORD
      selector: DETTargetSelector;  {target selector}
      aspectName: RStringPtr;       {aspect name}
      itemNumber: LongInt;          {sublist index number}
      dsSpec: PackedDSSpecPtr;      {DSSpec}
   END;
```

Forwarder List

```
TYPE
   DETForwarderListItem = RECORD
      next: ^DETForwarderListPtr;      {handle to next item, or nil}
      attributeValueTag: AttributeTag; {attribute value tag (0 for none)}
      rstrs: PackedPathName;           {forwarder list}
   END;

   DETForwarderListPtr = ^DETForwarderListItem;
   DETForwarderListHandle = ^DETForwarderListPtr;
```

Call Block Headers

```
TYPE
   DETCallBlockHeader =
   RECORD
      reqFunction: DETCallFunctions;{requested function}
      callBack: DETCallBack;        {pointer to callback routine}
```

```
      callBackPrivate: LongInt;     {private data for the callback routine}
      templatePrivate: LongInt;     {private data stored in template}
   END;

   DETCallBlockTargetedHeader =
   RECORD
      reqFunction: DETCallFunctions;{requested function}
      callBack: DETCallBack;        {pointer to callback routine}
      callBackPrivate: LongInt;     {private data for the callback routine}
      templatePrivate: LongInt;     {private data stored in template}
      instancePrivate: LongInt;     {private data stored in aspect}
      target: DETTargetSpecification;{the target (originator) of the call}
      targetIsMainAspect: Boolean;  {TRUE if the target is the main aspect}
   END;

   DETCallBlockPropertyHeader =
   RECORD
      reqFunction: DETCallFunctions;{requested function}
      callBack: DETCallBack;        {pointer to callback routine}
      callBackPrivate: LongInt;     {private data for the callback routine}
      templatePrivate: LongInt;     {private data stored in template}
      instancePrivate: LongInt;     {private data stored in aspect}
      target: DETTargetSpecification;{the target (originator) of the call}
      targetIsMainAspect: Boolean;  {TRUE if the target is the main aspect}
      property: Integer;            {the property number the call refers to}
   END;

   DETProtoCallBlock = DETCallBlockPropertyHeader;
```

Call Block Case Statement

```
TYPE
   DETCallBlock =
   RECORD
   CASE Integer OF
       1: (protoCall: DETProtoCallBlock);
       2: (init: DETInitBlock);
       3: (exit: DETExitBlock);
       4: (instanceInit: DETInstanceInitBlock);
       5: (instanceExit: DETInstanceExitBlock);
       6: (instanceIdle: DETInstanceIdleBlock);
       7: (propertyCommand: DETPropertyCommandBlock);
       8: (maximumTextLength: DETMaximumTextLengthBlock);
```

```
     9: (viewListChanged: DETViewListChangedBlock);
    10: (propertyDirtied: DETPropertyDirtiedBlock);
    11: (validateSave: DETValidateSaveBlock);
    12: (dropQuery: DETDropQueryBlock);
    13: (dropMeQuery: DETDropMeQueryBlock);
    14: (attributeCreationBlock: DETAttributeCreationBlock);
    15: (attributeNew: DETAttributeNewBlock);
    16: (attributeChange: DETAttributeChangeBlock);
    17: (attributeDelete: DETAttributeDeleteBlock);
    18: (itemNew: DETItemNewBlock);
    19: (patternIn: DETPatternInBlock);
    20: (patternOut: DETPatternOutBlock);
    21: (shouldSync: DETShouldSyncBlock);
    22: (doSync: DETDoSyncBlock);
    23: (openSelf: DETOpenSelfBlock);
    24: (convertToNumber: DETConvertToNumberBlock);
    25: (convertToRString: DETConvertToRStringBlock);
    26: (convertFromNumber: DETConvertFromNumberBlock);
    27: (convertFromRString: DETConvertFromRStringBlock);
    28: (customViewDraw: DETCustomViewDrawBlock);
    29: (customViewMouseDown: DETCustomViewMouseDownBlock);
    30: (keyPress: DETKeyPressBlock);
    31: (paste: DETPasteBlock);
    32: (customMenuSelected: DETCustomMenuSelectedBlock);
    33: (customMenuEnabled: DETCustomMenuEnabledBlock);
    34: (dynamicForwarders: DETDynamicForwardersBlock);
    35: (dynamicResource: DETDynamicResourceBlock);
END;

DETCallBlockPtr = ^DETCallBlock;
```

Callback Block Headers

```
TYPE
   DETCallBackBlockHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
   END;

   DETCallBackBlockTargetedHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
      target: DETTargetSpecification;     {the target for the request}
```

```
   END;

   DETCallBackBlockPropertyHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
      target: DETTargetSpecification;     {the target for the request}
      property: Integer;                  {the property to apply the
                                            request to}
   END;

   DETProtoCallBackBlock = DETCallBackBlockPropertyHeader;
```

Callback Block Case Statement

```
TYPE
   DETCallBackBlock =
   RECORD
   CASE Integer OF
      1: (protoCallBack: DETProtoCallBackBlock);
      2: (beep: DETBeepBlock);
      3: (busy: DETBusyBlock);
      4: (changeCallFors: DETChangeCallForsBlock);
      5: (getCommandSelectionCount: DETGetCommandSelectionCountBlock);
      6: (getCommandItemN: DETGetCommandItemNBlock);
      7: (getDSSpec: DETGetDSSpecBlock);
      8: (getTemplateFSSpec: DETGetTemplateFSSpecBlock);
      9: (getOpenEdit: DETGetOpenEditBlock);
      10: (closeEdit: DETCloseEditBlock);
      11: (getPropertyType: DETGetPropertyTypeBlock);
      12: (getPropertyNumber: DETGetPropertyNumberBlock);
      13: (getPropertyRString: DETGetPropertyRStringBlock);
      14: (getPropertyBinarySize: DETGetPropertyBinarySizeBlock);
      15: (getPropertyBinary: DETGetPropertyBinaryBlock);
      16: (getPropertyChanged: DETGetPropertyChangedBlock);
      17: (getPropertyEditable: DETGetPropertyEditableBlock);
      18: (setPropertyType: DETSetPropertyTypeBlock);
      19: (setPropertyNumber: DETSetPropertyNumberBlock);
      20: (setPropertyRString: DETSetPropertyRStringBlock);
      21: (setPropertyBinary: DETSetPropertyBinaryBlock);
      22: (setPropertyChanged: DETSetPropertyChangedBlock);
      23: (setPropertyEditable: DETSetPropertyEditableBlock);
      24: (dirtyProperty: DETDirtyPropertyBlock);
      25: (doPropertyCommand: DETDoPropertyCommandBlock);
```

```
      26: (sublistCount: DETSublistCountBlock);
      27: (selectedSublistCount: DETSelectedSublistCountBlock);
      28: (requestSync: DETRequestSyncBlock);
      29: (addMenu: DETAddMenuBlock);
      30: (removeMenu: DETRemoveMenuBlock);
      31: (menuItemRString: DETMenuItemRStringBlock);
      32: (openDSSpec: DETOpenDSSpecBlock);
      33: (aboutToTalk: DETAboutToTalkBlock);
      34: (breakAttribute: DETBreakAttributeBlock);
      35: (saveProperty: DETSavePropertyBlock);
      36: (getCustomViewUserReference: DETGetCustomViewUserReferenceBlock);
      37: (getCustomViewBounds: DETGetCustomViewBoundsBlock);
      38: (getResource: DETGetResourceBlock);
      39: (templateCounts: DETTemplateCounts);
      40: (unloadTemplates: DETUnloadTemplatesBlock);
   END;

   DETCallBackBlockPtr = ^DETCallBackBlock;
```

Functions You Can Provide as Part of Your Code Resource

Initializing and Removing Templates

```
TYPE
DETInitBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      newCallFors: LongInt;
   END;

DETExitBlock = DETCallBlockHeader;

DETInstanceInitBlock = DETCallBlockTargetedHeader;

DETItemNewBlock = DETCallBlockTargetedHeader;

DETInstanceExitBlock = DETCallBlockTargetedHeader;
```

Dynamic Creation of Resources

```
TYPE
DETDynamicForwardersBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      forwarders: DETForwarderListHandle;
   END;

DETDynamicResourceBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      resourceType: ResType;
      propertyNumber: Integer;
      resourceID: Integer;
      theResource: Handle;
   END;
```

Processing Idle-Time Tasks

```
TYPE
DETInstanceIdleBlock = DETCallBlockTargetedHeader;
```

Property and Information Page Routines

```
TYPE
DETOpenSelfBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
```

```
      targetIsMainAspect: Boolean;
      modifiers: Integer;
   END;

DETPropertyCommandBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      parameter: LongInt;
   END;

DETKeyPressBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theEvent: ^EventRecord;
   END;

DETPasteBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      modifiers: Integer;
   END;
```

```
DETMaximumTextLengthBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      maxSize: LongInt;
   END;

DETViewListChangedBlock = DETCallBlockTargetedHeader;

DETPropertyDirtiedBlock = DETCallBlockPropertyHeader;

DETValidateSaveBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      errorString: RStringHandle;
   END;
```

Supporting Drops

```
TYPE
DETDropMeQueryBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      modifiers: Integer;
      commandID: LongInt;
```

```
      destinationType: AttributeType;
      copyToHFS: Boolean;
   END;

DETDropQueryBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      modifiers: Integer;
      commandID: LongInt;
      destinationType: AttributeType;
      copyToHFS: Boolean;
   END;
```

Attribute-Related Commands

```
TYPE
DETAttributeCreationBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      value: Handle;
   END;

DETAttributeNewBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
```

```
      targetIsMainAspect: Boolean;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      value: Handle;
   END;

DETAttributeChangeBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      attrCID: AttributeCreationID;
      value: Handle;
   END;

DETAttributeDeleteBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      dsSpec: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
   END;
```

Processing Custom Lookup-Table Pattern Elements

```
TYPE
DETPatternInBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      elementType: LongInt;
      extra: LongInt;
      attribute: AttributePtr;
      dataOffset: LongInt;
      bitOffset: Integer;
   END;

DETPatternOutBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      elementType: LongInt;
      extra: LongInt;
      attribute: AttributePtr;
      data: Handle;
      dataOffset: LongInt;
      bitOffset: Integer;
   END;
```

Synchronizing Property Values

```
TYPE
DETShouldSyncBlock =
   RECORD
      reqFunction: DETCallFunctions;
```

```
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      shouldSync: Boolean;
   END;

DETDoSyncBlock = DETCallBlockTargetedHeader;
```

Custom Property-Type Conversions

```
TYPE
DETConvertToNumberBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: LongInt;
   END;

DETConvertToRStringBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      property: Integer;
      theValue: RStringHandle;
   END;

DETConvertFromNumberBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
```

```
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: LongInt;
   END;

DETConvertFromRStringBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: RStringPtr;
   END;
```

Custom Views and Custom Menus

```
TYPE
DETCustomViewDrawBlock = DETCallBlockPropertyHeader;

DETCustomViewMouseDownBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theEvent: ^EventRecord;
   END;

DETCustomMenuEnabledBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
```

```
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      menuTableParameter: Integer;
      enable: Boolean;
   END;

DETCustomMenuSelectedBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      menuTableParameter: Integer;
   END;
```

CE-Provided Functions That Your Code Resource Can Call

Calling CE-Provided Functions

```
{There is no Pascal equivalent to the C macro for calling callback routines.
```

Testing Your Code Resource

```
DETBeepBlock = DETCallBackBlockHeader;

DETBusyBlock = DETCallBackBlockHeader;

DETChangeCallForsBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      newCallFors: LongInt;
   END;

DETGetCommandSelectionCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      count: LongInt;
   END;
```

```
DETGetCommandItemNBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      itemNumber: LongInt;
      itemType: DETItemType;
      CASE Integer OF
         1: (fsInfo: ^DETFSInfoPtr);
         2: (ds: RECORD
            dsSpec: ^PackedDSSpecPtr;{
            refNum: Integer;
            identity: AuthIdentity;
            END);
         3: (dsSpec: ^PackedDSSpecPtr);
         4: (ltrSpec: LetterSpecHandle);
   END;

DETGetDSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      dsSpec: ^PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      isAlias: Boolean;
      isRecordRef: Boolean;
   END;

DETGetTemplateFSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      fsSpec: FSSpec;
      baseID: Integer;
      aspectTemplateNumber: LongInt;
   END;

DETGetOpenEditBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      viewProperty: Integer;
   END;
```

```
DETCloseEditBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
   END;

DETGetPropertyTypeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyType: Integer;
   END;

DETGetPropertyNumberBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyValue: LongInt;
   END;

DETGetPropertyRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyValue: RStringHandle;
   END;

DETGetPropertyBinarySizeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyBinarySize: LongInt;
   END;

DETGetPropertyBinaryBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
```

```
      property: Integer;
      propertyValue: Handle;
   END;

DETGetPropertyChangedBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyChanged: Boolean;
   END;

DETGetPropertyEditableBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyEditable: Boolean;
   END;

DETSetPropertyTypeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newType: Integer;
   END;

DETSetPropertyNumberBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: LongInt;
   END;

DETSetPropertyRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: RStringPtr;
   END;
```

```
DETSetPropertyBinaryBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: Ptr;
      newValueSize: LongInt;
   END;

DETSetPropertyChangedBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyChanged: Boolean;
   END;

DETSetPropertyEditableBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyEditable: Boolean;
   END;

DETDirtyPropertyBlock = DETCallBackBlockPropertyHeader;

DETDoPropertyCommandBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      parameter: LongInt;
   END;

DETSublistCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      count: LongInt;
   END;

DETSelectedSublistCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
```

```
      target: DETTargetSpecification;
      count: LongInt;
   END;

DETRequestSyncBlock =  DETCallBackBlockTargetedHeader;

DETAddMenuBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      name: ^RString;
      parameter: LongInt;
      addAfter: LongInt;
   END;

DETRemoveMenuBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      itemToRemove: LongInt;
   END;

DETMenuItemRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      itemParameter: LongInt;
      rString: RStringHandle;
   END;

DETOpenDSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      dsSpec: PackedDSSpecPtr;
   END;

DETAboutToTalkBlock = DETCallBackBlockHeader;

DETBreakAttributeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
```

```
      breakAttribute: AttributePtr;
      isChangeable: Boolean;
   END;

DETSavePropertyBlock = DETCallBackBlockPropertyHeader;

DETGetCustomViewUserReferenceBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      userReference: Integer;
   END;

DETGetCustomViewBoundsBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      bounds: Rect;
   END;

DETGetResourceBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      resourceType: ResType;
      theResource: Handle;
   END;

DETTemplateCounts =
   RECORD
      reqFunction: DETCallBackFunctions;
      aspectTemplateCount: LongInt;
      infoPageTemplateCount: LongInt;
   END;

DETUnloadTemplatesBlock = DETCallBackBlockHeader;
```

Result Codes

Result codes in the range of –15000 to –15039 are reserved for AOCE templates.

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETInvalidCommandItemNumber`	–15007	Command item number out of range
`kDETUnableToGetCommandItemSpec`	–15008	Unable to retrieve information about item (possibly out of memory)
`kDETRequestedTypeUnavailable`	–15009	Item could not be represented in the specified format
`kDETInvalidDSSpec`	–15010	Could not resolve `DSSpec`
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open
`kDETNoSuchView`	–15013	No view found with specified property number
`kDETCouldNotAddMenuItem`	–15014	Could not add item to menu
`kDETCouldNotRemoveMenuItem`	–15015	Could not remove item from dynamic menu
`kDETCouldNotFindMenuItem`	–15016	Could not find menu item
`kDETCouldNotFindCustomView`	–15017	Could not find custom view
`kDETInvalidReqFunction`	–15018	Invalid callback routine selector
`kDETInvalidCallBack`	–15019	Invalid callback (for reasons other than those above)
`kDETPropertyBusy`	–15020	Specified property is being edited